VISUALIZATION OF SORTING ALGORITHMS

By

VIKAS MUKTAVARAM

Bachelor of Technology

Osmania University College of Technology

Hyderabad, India

1994

Submitted to the faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December 1996

VISUALIZATION OF SORTING ALGORITHMS

Thesis Approved:

Mansur Samadzadeh

Thesis Advisor

J Chandler

Blayne E. Mayfield

Thomas C. Collins

Dean of the Graduate College

PREFACE

The purpose of this thesis was to develop a graphical tool for the visualization of a number of sorting methods. The tool developed, called SortDisplay, not only gives the users the option to view various sorting methods in execution, but also gives other options to get information about the performance and complexity of the sorting algorithms in terms of the number of comparisons and exchanges needed. The tool also has various other options which help the user understand and analyze the sorting algorithms implemented. The tool was designed to be an educational tool running on the Oklahoma State University Computer Science Department's Sequent Symmetry S/81 computer running the Dynix/ptx operating system. The following topics, which were covered as background and context, put the thesis work in perspective: (1) sorting and sorting algorithms; (2) the concept of visualization and types of visualization; (3) the X window system, the X protocol, and various software layers in X; (4) the OSF/Motif toolkit; and (5) using Motif with C++.

The programming part of the tool involved designing and implementing the class hierarchies, application framework, sorting methods, and user-interface. The program, coded in the C++ programming language using the Motif toolkit, Xt Intrinsics, and Xlib, has over 4500 lines of uncommented code (over 5500 lines of documented code), 2 major class hierarchies, and 40 classes. The six sorting methods that were implemented include two Insertion sorts (Linear Insertion and Shellsort), three Exchange sorts (Bubblesort, Combsort,

and Quicksort), and one Exchange sort (Straight Selection). SortDisplay was evaluated by various users of the Computer Science Department's Sequent Symmetry S/81 machine including faculty and staff members, and former and current graduate students. Most of the recommendations and suggestions of the evaluators were incorporated into the tool. A number of the ideas that were deemed beyond the scope of the present work were left for the future updates of the tool.

ACKNOWLEDGMENTS

First of all, I would like to thank my Guru Dr. Mansur Samadzadeh for his ideas, advice and encouragement. He truly has been an inspiration to me and brought out the best in me. I would also like to express my gratitude and appreciation to Drs. Blayne Mayfield and John Chandler for serving on my Graduate committee and for providing valuable suggestions and ideas.

I'm grateful to all the people who evaluated the tool and provided useful feedback which helped me improve the tool. The tool is a better one for their comments and suggestions.

I cannot thank enough all my friends who have been friends indeed! And finally, my parents and brother - it's their love that keeps me going.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

Figure Page

Figure Page

CHAPTER I

INTRODUCTION

Visualization is the process of graphically or pictorially representing objects, concepts, or processes. Visualization is attention-grabbing, and generally more efficient than the verbal or numerical presentation of the same information or data in terms of understandability and user-friendliness. Graphical representation of data helps in extracting, abstracting, and presenting meaningful, pertinent information from large volumes of complex data. Visual demonstrations and interactive visualization tools can be used effectively to convey complex ideas.

Graphical display of sorting methods dynamically while in execution would help in better understanding and comparative appreciation of the sorting algorithms. To have such a tool on the Oklahoma State University Computer Science Department's Sequent Symmetry S/81 would be helpful pedagogically. The primary objective of this thesis was to develop a graphical tool for the visualization of a number of sorting methods. The sorts considered here include Linear Insertion, Shellsort, Bubblesort, Combsort, Quicksort, and Straight Selection.

The tool, called SortDisplay, was implemented using the Motif Toolkit on the Sequent S/81 computer running the DYNIX/ptx operating system. Chapter II of this thesis discusses the concept of visualization and its types, sorting and sorting algorithms, and the

visualization of sorting algorithms. Chapter III describes the implementation platform and environment in terms of the toolkit and the programming language used. Chapter IV takes a look at the design and implementation issues involved in developing SortDisplay. The testing and evaluation of SortDisplay are discussed in Chapter V. Finally, Chapter VI summarizes the thesis work and provides some suggestions for future work.

CHAPTER II

VISUALIZATION AND SORTING ALGORITHMS

2.1 Visualization

Patterns and shapes are generally less abstract than numbers and words. Information that is presented graphically (as pictures and graphs) conveys more readily and permits better retention and assimilation than the verbal or textual representation of the same information. Graphics have been used for centuries to communicate information effectively among people and aid in the comprehension of complex information [House 82]. Visualization can be described in Gershon's words as "the process of transforming information into a visual form, enabling users to observe the information" [Gershon 92].

2.1.1 Types of Visualization

Visualization could be broadly classified into Data Visualization, Algorithm Animation, and Program Visualization.

- Data Visualization refers to the graphical representation of application-oriented data. Gershon observes that "data represented in a visual form is used for visual analysis of data and for scanning data for existence of desired features" [Gershon 92].
- Algorithm Animation refers to the abstraction of a program's data, operations, and semantics, and the creation of dynamic graphical views of those abstractions [Gerald and

Ueberhuber 94]. Gerald and Ueberhuber gave the following definition for Algorithm Animation.

> Algorithm animation includes the exposition of program properties by displaying multiple dynamic views of the program and associated data structures. It also encompasses program animation and data-structure rendering, which typically involve one-to-one mappings between program data and animation images. However, algorithm animation is broader than these two areas as it involves program views that go beyond simple data-structure presentation.

- Program Visualization refers to the pictorial representation of different aspects of a program which is specified in a conventional, textual manner [Gerald and Ueberhuber 94].

2.2 Sorting and Sorts

Sorting could be defined as the ordering of elements of a set according to some predefined rules of order. Knuth defines sorting as "the rearrangement of items into ascending or descending order" [Knuth 73].

This section gives an overview of a number of sorting algorithms. The sorts could be grouped (based on the way data is compared and moved while sorting) as:

Insertion Sorts: Linear Insertion; Binary Insertion; and Shellsort.
Exchange Sorts: Bubblesort; Combsort; Shakersort; and Quicksort.
Selection Sorts: Straight Selection; Tree Selection; and Heapsort.

Some of the other sorts include Sorting by Merging and Sorting by Distribution [Knuth 73].

The ensuing discussion is based on the following assumptions and definitions.

1. Sorting is arranging a sequence of nonnegative integers in a non-decreasing order.

2. The sorts considered here are internal sorts, i.e., the entire sorting can be done in main memory. In other words, the number of elements to be sorted is relatively small (small

enough to fit in the internal memory). Sorts that cannot be performed in main memory and must be done on disk or tapes are called external sorts [Knuth 73] and are not considered here.

3. The calculation of the running time of a sorting algorithm is based on the number of algorithm steps, i.e., the number of comparisons and exchanges, required to sort n items. As Sedgewick points out, "sorting programs access records in one of two ways: either keys are accessed for comparison, or entire records are accessed to be moved" [Sedgewick 90]. It is assumed that the implementation of sorting algorithms here "avoids shuffling the records around by doing an indirect sort, i.e., the records themselves are not necessarily rearranged, but rather an array of pointers (or indices) is rearranged so that the first pointer points to the smallest record, etc." [Sedgewick 90].

4. Another important factor to be considered is the space complexity of the sorting algorithms. All the algorithms require an array of size n to hold the n records to be sorted. The amount of extra memory required is an important factor to be considered. The sorting methods could be classified into three types [Sedgewick 90] on the basis of extra memory required: a) those that sort in place and use no extra memory, except perhaps for a small stack or table; b) those that use a linked-list representation and thus use n extra words of memory for list pointers; and c) those that need enough extra memory to hold another copy of the array to be sorted.

5. The following definitions apply [Weiss 93].

'O': $T(n) = O(f(n))$ if there are constants c and n_0 such that $T(n) \leq c\ f(n)$ when $n \geq n_0$. 'O' implies that the growth rate of T(n) is less than or equal to that of f(n).

'Ω': $T(n) = \Omega(g(n))$ if there are constants c and n_0 such that $T(n) \geq c\ g(n)$ when $n \geq n_0$. 'Ω' implies that the growth rate of T(n) is greater than or equal to that of g(n).

'θ': $T(n) = \theta(h(n))$ if and only if $T(n) = O(h(n))$ and $T(n) = \Omega(h(n))$. 'θ' implies that the growth the rate of T(n) is equal to the growth rate of h(n).

The following subsections contain a brief discussion of the sorting algorithms that were considered for this thesis. These sorting methods were selected based on the ease of representing the sorting process graphically in terms of data comparisons and movements.

2.2.1 Linear Insertion

This is one of the simplest sorting algorithms. This algorithm could be compared with the task of picking up cards one by one for a bridge hand. Each new card is inserted into the correct position, relative to the other cards already considered in the hand [Sorting out Sorting 81]. The sort consists of n - 1 passes for n elements to be sorted. For passes p = 2 through n, the sort ensures that the elements in positions 1 through p are in sorted order. It makes use of the fact that elements in positions 1 through p - 1 are already known to be in sorted order [Weiss 93].

The following linear insertion sort algorithm is based on the one given by Knuth [Knuth 73]. Let $R_1, R_2, \ldots, R_n$ represent the records to be sorted. Each record holds one key. After sorting, the records' keys will be in the order $K_1 \leq K_2 \leq \ldots \leq K_n$.

1. [Loop on j] Perform Steps 2 through 5 for $j = 2, 3, \ldots, n$, then terminate the algorithm.
2. [Set up i, K, R] Set $i = j - 1$, $K = K_j$, and $R = R_j$.
3. [Compare K, K_i] If $K \geq K_i$, go to Step 5.
4. [Move R_i, decrease i] Set $R_{i+1} = R_i$, then $i = i - 1$. If $i > 0$, go to Step 3.

5. [R into R_{i+1}] Set $R_{i+1} = R$.

This sort, according to Sedgewick "uses about $n^2/2$ comparisons and $n^2/8$ data movements on the average, and twice as many in the worst case" [Sedgewick 90]. So the algorithm has a running time complexity of $O(n^2)$, where n is the number of records to be sorted. This bound is tight because input in reverse can also achieve this bound. So, the average case and worst case time complexity is $\Omega(n^2)$. If the input is pre-sorted, the running time would be O(n) [Weiss 93]. Since sorting is done in-place, no extra memory is needed. So the space complexity of the algorithm is O(n), for n records.

2.2.2 Shellsort

Shellsort is named after its inventor, Donald Shell [Shell 59]. It was one of the first algorithms to break the quadratic time barrier. The first algorithm to improve on $O(n^2)$ was mergesort, suggested as early as 1945 by John von Neumann [Knuth 73]. It works by making comparisons between elements that are distant; the distance between comparisons decreases as the algorithm runs, and in the last phase, adjacent elements are compared. For this reason, Shellsort is sometimes referred to as the diminishing increment sort.

Shellsort uses a sequence, $h_1, h_2, \ldots, h_t$ called the increment sequence. Although any choice of increment sequence will do as long as $h_1 = 1$, some choices are better than others in terms of the number of comparisons and exchanges needed.

The following algorithm is based on one given by Knuth [Knuth 73]. Records R_1, $R_2 \ldots, R_n$ are to be sorted in the ascending order of their keys. Each record holds one key. After sorting, the keys will be in the order $K_1 \le K_2 \le \ldots \le K_n$.

1. [Loop on s] Perform Step 2 for s = t, t - 1, . . . , 1, then terminate the algorithm.

2. [Loop on j] Set $h = h_s$ and perform Steps 3 through 6 for $h < j \leq n$.

3. [Set up i, K, R] Set $i < j - h$, $K = K_j$, and $R = R_j$.

4. [Compare K, K_i] If $K \geq K_i$, go to Step 6.

5. [Move R_i, decrease i] Set $R_{i+h} = R_i$, then $i = i - h$. If $i > 0$, go to Step 4.

6. [R into R_{i+h}] Set $R_{i+h} = R$.

One increment choice for Shellsort is Shell's increment sequence [Knuth 73]:

$h_t = \lfloor h/2 \rfloor$ and $h_k = \lfloor h_{k+1} \rfloor$

The worst case running time of Shellsort using Shell's increment sequence is $\Theta(n^2)$, where n is the number of records to be sorted. A popular increment sequence used is Hibbard's increment sequence [Knuth 73]:

$1, 3, 7, \ldots, 2^k-1$, for $k = 1, 2, \ldots, n$.

The worst case running time of Shellsort using Hibbard's increment sequence is $\Theta(n^{3/2})$ [Weiss 91]. The average case time is not known. Since sorting is done in-place, no extra memory is needed. So the space complexity of the algorithm is O(n), where n is the number of records to be sorted.

2.2.3 Bubblesort

The method is called "bubble sorting" because large elements "bubble up" to their proper position. Knuth observed that "perhaps the most obvious way to sort by exchanges is to compare adjacent keys and swap the items if they are out of order" [Knuth 73]. Bubblesort is also known as exchange selection or propagation sort.

The following algorithm is based on one given by Knuth [Knuth 73]. Records R_1, $R_2, \ldots, R_n$ are to be sorted in ascending order. Each record holds one key. After sorting is

completed, the keys will be in the order $K_1 \leq K_2 \leq \ldots \leq K_n$.

1. [Initialize BOUND] Set BOUND = n.

2. [Loop on j] Set t = 0. Perform Step 3 for j = 1, 2, . . . , BOUND - 1, and then go to Step 4.

3. [Compare and Swap] If $K_j > K_{j+1}$, swap R_j and R_{j+1}, and set t = j.

4. [Any more exchanges?] If t = 0, the algorithm terminates. Otherwise, set BOUND = t and return to Step 2.

As far as time complexity calculated in terms of the number of critical operations is concerned, according to Sedgewick "bubblesort uses about $n^2/2$ comparisons and $n^2/2$ exchanges on the average and in the worst case" [Sedgewick 90]. So the algorithm has a running time complexity of $\Omega(n^2)$, where n is the number of records to be sorted. If the input is pre-sorted, no swaps are ever needed and Step 3 is executed n times [Aho et al. 83]. So the running time complexity would be O(n). Sorting is done in-place and so no extra memory is required. So the space complexity of the algorithm is O(n), for n records to be sorted.

2.2.4 Combsort

Combsort was introduced by Richard Box and Stephen Lacey [Box and Lacey 91]. It is simple and yet efficient [Su 93]. It has been developed based on bubblesort. Bubblesort, as described in the previous subsection, is one of the simplest algorithms used for sorting. It is easy to program and debug and also consumes little extra memory [Box and Lacey 91]. But bubblesort is slow for most lists of data. A few simple modifications to the original bubblesort routine can make it a fast and efficient sort for all kinds of lists [Box and Lacey 91]. The modified sorting method is called Combsort.

Bubblesort works by comparing each element to the next and swapping the elements if they are out of order. A bubblesort is done when it makes a pass that doesn't require any swaps. Bubblesort is slow because it is susceptible to the birth of elements called turtles, "a turtle is a relatively low value located near the end of a list when the list is to be sorted in ascending order" [Box and Lacey 91]. Bubblesort is modified to eliminate turtles by allowing the gap (defined as the distance between compared elements) to be greater than 1, as opposed to the original bubblesort in which the gap is always 1. This minor change makes Combsort efficient while still retaining the simplicity of bubblesort.

Initially, the gap is the list length divided by 1.3, which is called the shrink factor [Box and Lacey 91]. The shrinkfactor of 1.3 was determined empirically to be the ideal gap by testing Combsort over 200,000 random lists [Box and Lacey 91]. Before each subsequent pass, the gap is reduced to the value of the previous gap divided by 1.3; and if this quotient becomes less than 1, it is reset to 1. Combsort, therefore, uses a sequence of diminishing increments, as does Shellsort.

Another improvement added to improve Combsort is the Rule of 11, with the new sort being called Combsort11. The Rule of 11 eliminates gap sizes of 9 and 10 and always includes 11.

The following algorithm for Combsort11 is based on the C program given by Box and Lacey [Box and Lacey 91].

1. [Initialize switches] Initialize switches = 0.
2. [Set gap] Set gap = size of list.
3. [Loop] Perform Steps 4 through 9 while gap > 1 or switches > 0.

4. [Reset gap] Set gap = gap / shrink factor. If gap = 0 , reset gap = 1; else if gap = 9 or 10, reset gap = 11.

5. [Reset switches] Set switches = 0.

6. [Set top] Set top = size - gap.

7. [Inner loop] Perform Steps 8 and 9 for i = 0, . . . , top - 1.

8. [Set j] Set j = i + gap.

9. [Compare, Swap, and Update] If j^{th} element > i^{th} element, then swap i^{th} and j^{th} elements and switches = switches + 1.

The empirical results given by Box and Lacey [Box and Lacey 91] and Su [Su 93] indicate that the average case running time complexity of the algorithm for sorting random lists is comparable to that of Heapsort and Quicksort. So the running time of the algorithm appears to be O(n log n). Also, Combsort sorts a sorted list and a reverse sorted list faster than it would sort a random list [Su 93]. Since sorting is done in-place, no extra memory is required. So the space complexity of the algorithm is O(n), where n is the number of records to be sorted.

2.2.5 Quicksort

As the name implies, Quicksort is the fastest known sorting algorithm in practice. Quicksort is a divide-and-conquer recursive algorithm. Weiss gives the basic algorithm to sort an array S of n elements in the following four steps [Weiss 93].

1. If the number of elements in S is 0 or 1, then return.

2. Pick any element v in S. This is called the pivot.

3. Partition S - {v} (the remaining elements in S) into two disjoint groups, one less than v and the other greater than v:

   $S_1 = \{x \in S - \{v\} \mid x < v\}$ and $S_2 = \{x \in S - \{v\} \mid x > v\}$.

4. Return {quicksort(S_1), followed by v, followed by quicksort(S_2)}.

The reason why Quicksort is fast is that the partitioning step can actually be performed in place and very efficiently. Its average running time is O(n log n), where n is the number of records to be sorted. It is fast mainly due to a tight and optimized inner loop. Quicksort cannot perform as well as expected unless the choice of the pivot is good. If the pivot is chosen very close to one side of the sequence, the running time will be higher. For example, if the pivot is the smallest element in the sequence, the partition requires n-1 comparisons, resulting in only placing the pivot in the right position. If the sequence is already in non-decreasing order and the first element is always selected as the pivot, the running time of the algorithm is $O(n^2)$ [Manber 89].

The quadratic worst case can be eliminated for sequences that are in sorted or almost sorted order by comparing the first, last, and middle elements, and then taking the median of these three, i.e., the second largest, as the pivot. This method of choosing the partition is called the Median-of-Three partitioning [Weiss 93]. An even safer method is to choose pivots from among the elements in the sequence at random. The running time of Quicksort will still be $O(n^2)$ in the worst case, because there is still a chance that the pivot is the smallest element in the sequence. However, the likelihood that this worst case will occur is small [Manber 89]. Sorting is done in-place and in the average case uses only a small auxiliary stack to handle recursion. So the average case space complexity for the algorithm is O(n), where n is the number of records to be sorted. However, in the worst case, when the array is already sorted, the extra space required to handle recursion will be about the size of the number of records to be sorted [Sedgewick 90]. As mentioned above, the likelihood of

this worst case occurring is small.

2.2.6 Straight Selection

This is probably one of the simplest sorting algorithms to understand [Sorting out Sorting 81]. While sorting, an item is moved to the beginning of the array by exchanging it with the item currently in the desired location.

The following algorithm is based on the one given by Knuth [Knuth 73]. Records $R_1, R_2, \ldots, R_n$ are to be sorted. Each record holds one key. After sorting their keys will be in the order $K_1 \leq K_2 \leq \ldots \leq K_n$.

1. [Loop on j] Perform Steps 2 and 3 for j = n, n - 1, . . . , 2.
2. [Find the maximum key] Search through $K_j, K_{j-1}, \ldots, K_1$ to find the maximum key and let it be K_i.
3. [Exchange with R_j] Swap records R_i and R_j.

The sort uses about $n^2/2$ comparisons and n exchanges to sort an array of n elements [Sedgewick 90]. Regardless of the amount of order already existing in the input data, the algorithm performs Steps 2 and 3, n(n -1)/2 times, so the worst case and average case time complexity is $\Omega(n^2)$, where n is the number of records to be sorted [Aho et al. 83]. Since sorting is done in-place and no extra memory is required, the space complexity of the algorithm is O(n), for n records.

2.3 Related Work

Because of the importance of visualization and animation in the understanding of algorithms, animation of sorting algorithms has been implemented on different platforms using different tools and techniques. Three related implementations are discussed here.

Ronald Baecker's well-known work *Sorting out Sorting* [Sorting out Sorting 81] is an excellent algorithm animation film that shows visualization of nine sorting algorithms. The film is useful and engaging because of good visualization, animation, and a narrative that accompanies it. The film explains each sorting method in some detail and also compares the performance of all the nine sorting methods on sample data.

XTANGO is a general purpose algorithm animation system that supports programmers' developing smooth animation of their own programs. Sorting algorithms have been implemented in the XTANGO environment using rectangles, circles, etc. to represent the data. An http site that has animations for some of the sorting methods is www.cc.gatech.edu/gvu/softviz/algoanim/xtango.html.

Microsoft Visual C++ v1.51 software has a demonstration program that implements animation of some of the sorting methods. This program also uses sticks to represent sample data. The program has options to vary the speed of display and add sound. The program also gives comparative statistics for each run.

CHAPTER III

IMPLEMENTATION PLATFORM AND ENVIRONMENT

3.1 Implementation Platform

The Sequent Symmetry S/81, is a mainframe class computer system with a multi-processor architecture. The multi-processing and shared memory architecture consists of the following elements [Sequent 90].

- A parallel architecture that utilizes multiple industry-standard microprocessors.
- Either the DYNIX v3.0 operating system or the DYNIX/ptx operating system, both UNIX ports.
- A standard set of network interfaces such as Ethernet, SCSI, VMEbus, and MULTIBUS.

The Sequent Symmetry S/81's operating system has been engineered to incorporate features that support the underlying parallel architecture. Software that has been written for the UNIX operating system can run on the Sequent Symmetry S/81 with little or no modification. In the case of multi-user applications, the operating system of the Sequent Symmetry S/81 automatically distributes the tasks to multiple processors in an attempt to reduce response time and increase system throughput [Sequent 90].

The DYNIX v3.0 operating system supports the two major command sets of UNIX, namely the Berkeley UNIX and UNIX System V. On the other hand, the DYNIX/ptx operating system is compatible with AT&T System V v3.2 only [Sequent 90].

3.2 Implementation Environment

Research has shown that there can be substantial differences between the degree of user-friendliness of a program and the quality of its interface, in learning time, performance speed, error rates, and user satisfaction [Shneiderman 91]. One solution to the problem is to use graphical user interfaces (GUI's) and graphical display, which can help the user use the commands more easily and assimilate the data more readily.

The X Window System™ provides an "open" window system that supports a broad and powerful graphical windowing system foundation with very few restrictions on the development of high level applications. The X Window System™ supports mechanisms for implementing graphical user interfaces (GUI's) without imposing a particular user-interface style [Smith 91].

3.2.1 X Window System™

Nye gives the following introduction to X [Nye 90].

> The X window system (or simply X) provides a hierarchy of resizable windows and supports high performance device independent graphics. Unlike most other window systems for UNIX that have a built-in user interface, X is a substrate on which almost any style of user-interface can be built. But what is most unusual about X is that it is based on an asynchronous network protocol rather than on procedure or system calls. The advantage of using this protocol basis is that both local and network applications can be operated in the same way.

X is highly portable and as Nye observes, it is "so hardware independent and operating system independent that properly written application software will compile and run on any system" [Nye 90]. X differs from other widow systems in that it is not one homogeneous piece of software but has three interrelated parts: a server, a client, and a

network layer [Mansfield 91]. The server is the software that manages the display, keyboard, and mouse. The client is a program displaying on the screen and taking input from the keyboard and mouse. The client and server interact by the client sending drawing and information requests to the server and the server responding with user input, information, or error reports. The client and server may be running on the same machine or on different machines.

3.2.2 Network - The X Protocol

The X window system is defined by Nye as follows [Nye 90].

> The X protocol is the true definition of the X window system. Below the X protocol any lower level of networking can be used, as long as it is bi-directional and delivers bytes in sequence and unduplicated between a server process and a client process.

If the client and server are on the same machine, the network protocol could be using the local interprocess communication (IPC) channels. If they are running on different host machines, the network protocol could be using the Internet protocol using TCP/IP or DECnet. The protocol is designed to be operated asynchronously since this allows much higher performance.

3.2.3 Software Layers

The hierarchy of different layers that constitute X is given in Figure 1. As can be seen, the X protocol is the lowest level. It can be considered the machine language of X [Barkakati 91]. Clients implement the X protocol via a programming library that interfaces to a single underlying network protocol [Nye 90].

Xlib, which is the next higher level in the hierarchy, can be considered the assembly language of X [Barkakati 91]. It is a C language-based client programming library that uses

sockets on systems based on Berkeley UNIX, and provides code which simulates sockets' interfaces on systems based on AT&T's UNIX system V [Nye 90]. Although, X provides access to the X protocol through more than three hundred routines [Barkakati 91] and gives the programmer a great deal of control over X applications, its capabilities are basic. It would take a lot of effort using a lot of Xlib routines to develop a graphical user interface for any application.

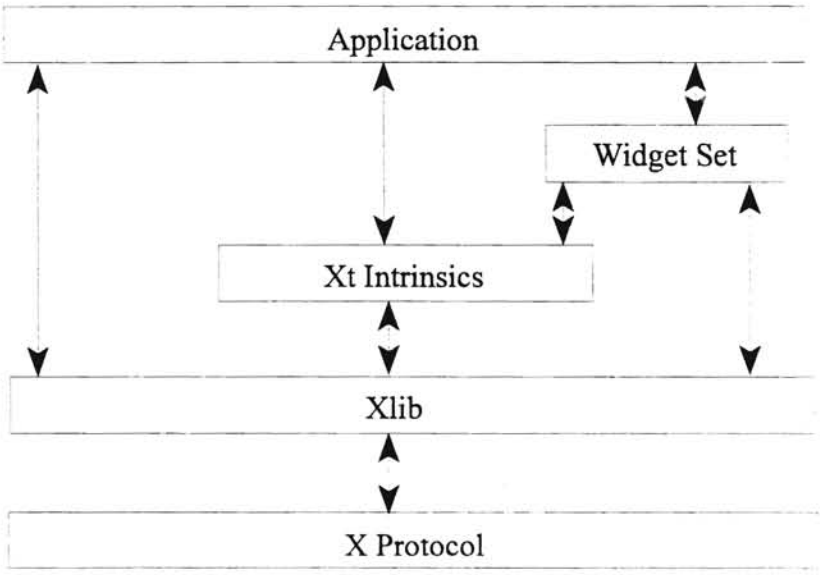

Figure 1. Hierarchy of software layers of X (source: [Smith 91])

At the next higher level is the X Toolkit Intrinsics (also called Xt Intrinsics), which can be considered the high level language of X [Barkakati 91]. Xt Intrinsics takes an object oriented approach to implementing basic building blocks known as widgets [Du 93]. A widget could be defined as "a collection of data structures encapsulated with a public functional interface that implements a high-level GUI component such as a menu" [Smith 91]. Xt Intrinsics provides functions applicable to all widgets and is the basis for other

toolkits like OSF/Motif.

3.2.4 OSF/Motif

OSF/Motif is a toolkit developed by the Open Software Foundation [Heller 91]. The OSF/Motif Toolkit is a set of functions and procedures that provide quick and easy access to the lower layers of the X window system [Dabbi 94]. The OSF/Motif functions and procedures provide user-interface objects known as widgets. OSF/Motif is a specification rather than an implementation, which makes it entirely implementation independent [Heller 91]. The primary window functions such as resizing, closing, moving, and iconizing are handled by the Motif Window Manager (MWM). The architecture of OSF/Motif Toolkit is shown in Figure 2 [Berlage 91].

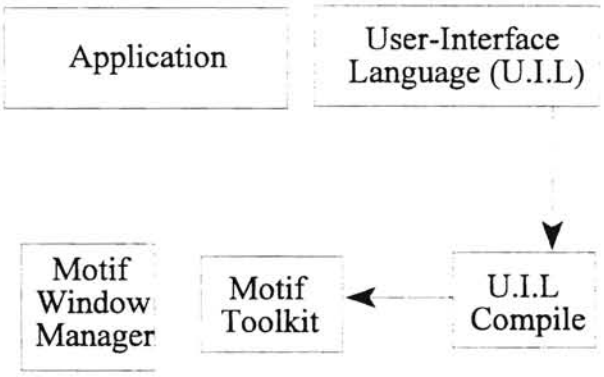

Figure 2. Architecture of OSF/Motif (source: [Berlage 91])

Motif provides a set of basic components that can be assembled into a graphical user interface. It provides the user interface components with which the user interacts. It also provides widgets that can be used for the more visible application controls such as push buttons, menus, labels, dialog boxes, scroll bars, and text entry or display areas.

3.2.5 Using Motif with C++

As mentioned earlier, Motif is based on the Xt Intrinsics, a library that supports an object-oriented architecture implemented in C. C++ is a language derived from C but which provides direct support for object oriented programming. There is a catch though: the objects supported by Xt and Motif are completely different from C++ objects [Young 95]. On the other hand, by using C++ with the Motif toolkit, both of which provide an abstraction layer that can be generalized for reusability, code that is abstract and generalized can be produced [Bernstein 95].

The general design goal of large software development projects is to increase reusability and improve productivity by reducing the amount of code a programmer must write. There are at least two ways to develop a point and click application to reduce the amount of code one must write. The traditional approach is to provide collections of functions or classes that implement common components needed by many programs. Motif is a typical example of a toolkit based on this approach. Motif provides a set of ready-to-use user-interface components that can be used with one another effectively to form a new application. Toolkits are effective and widely used forms of reusable software [Young 95].

Another approach, as defined by Young [Young 95], is to pay less attention to individual components needed by various applications, and to focus instead on the structure and control flow within a particular type of application. Here, the goal is to spare the programmer from having to define the architecture of each new application. An application framework provides a way to capture the characteristics, particularly the organizational characteristics, common to many applications. Analogous to a toolkit, an application

framework is a library that provides various components needed by programs. However, unlike traditional toolkits, an application framework also defines most of the connections between these components and defines the overall structure of applications built on the framework [Young 95].

CHAPTER IV

DESIGN AND IMPLEMENTATION

This chapter examines the design and implementation issues involved in developing the SortDisplay program including the class hierarchies, the application framework, sorting, dynamic display, and other related issues. The program has been implemented in the C++ programming language, following C++ and object-oriented programming and design conventions. The program has 40 classes and each class is declared in a file class.h (class is the name of the class) and implemented in class.C.

4.1 Overview of the Tool

The SortDisplay program enables users to observe the dynamic display of sorting methods. It also allows users to get current statistics, performance and complexity analysis graphs, and other useful information about the sorting algorithms.

Initially, the user selects a file that contains unsorted data. Until the user selects a valid data file, the "Sort" menu is not activated. After the user selects a valid data file, the unsorted data is displayed in a window titled "Unsorted Data". The numbers are represented by rectangles whose length is sized proportional to the magnitude of the values they represent. Now the "Sort" menu becomes active. Any number of the six sorting methods can be selected. Corresponding to each sorting method selected, the data is displayed again

in a window titled by the name of the sorting method. Using the "Display Sorting" command, all of the sorting methods selected can be started at the same time. The sorting display windows now show the dynamic display of sorting methods in execution. The two numbers that are compared or swapped at each step are represented by the darkening of the rectangles that represent the numbers. The number of comparisons and swaps that have been made are also displayed dynamically in the window. The "Reset" command can be used at any time to restart the sorting.

The speed of sorting display can be varied using the "Speed" command. Pushing the "Statistics" command button causes a window to pop up showing the number of comparisons and the number of swaps each sorting method took to sort the given data. The options under the "Info" menu provide useful information about the sorting algorithms. There is also a "Help" menu which provides information on running the program. And finally, the "Exit" command under the "File" menu can be used to exit the program.

4.2 Sorting and Display

There are two main aspects to the SortDisplay program: one is to sort the data and store the sorting information, i.e., the two numbers that are compared and the numbers that are moved around at every step of the sorting process; and the other is the dynamic display of the sorting process.

4.2.1 Sorting

Sorting is implemented using the abstract class Sort which supports an absolute virtual member function doTheSorting() which is implemented in the derived classes.

Unsorted data is stored and accessed using the class UnsortedData and the sorting information using the class SortInfo. The sorting class hierarchy is shown in Figure 3.

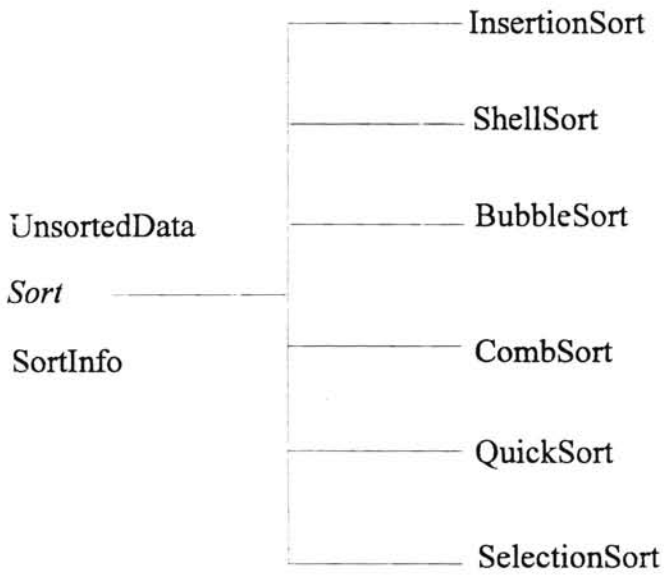

Figure 3. Sorting Class Hierarchy

4.2.2 Dynamic Display

The data is represented as rectangles whose lengths are sized according to the magnitude of the values they represent, and the sorting process is shown by darkening the rectangles that represent the numbers that are being compared or swapped at each state of the sorting process. The drawing is done on a canvas, which is a pixmap but which doesn't appear on the screen until copied.

Dynamic display of the sorting process is implemented using a technique called double-buffering. Double-buffering is a technique that uses two drawing canvases and at any given time, one of these canvases is visible, while the other remains hidden. All objects that appear on the next frame of the animation are drawn on the hidden canvas. When the

entire scene has been rendered, the entire canvas is displayed at once. The previously visible canvas is hidden, erased, and made available for drawing the next frame. The result is a smooth animation [Young 95].

The Canvas class provides the pixmap and also supports other drawing functions such as drawCircle(), drawRectangle(), drawLine(), and drawString(). The class's public protocol also includes a function to copy the drawing onto the screen, copy(), and functions to clear the pixmap with black background, blacken(), and with white background, whiten(). The windows which provide the drawing area to display the pixmaps are described later in this chapter.

4.3 User-Interface

4.3.1 Initialization

The initialization of the SortDisplay application program is handled by the Application class which does all of the following.

- Initialize the Xt Intrinsics.
- Open a connection to the X server.
- Handle events by entering an event loop.

Apart from the initialization, the Application class handles additional responsibilities including:

- Maintaining global data structures including the X display and application context.
- Creating a main shell that serves as a parent for all top-level windows and dialogs.

Since the application must create a single instance of the Application class (which can be accessed throughout the program) a global instance, theApplication is declared, which

is instantiated in the main routine.

4.3.2 Application Framework

The design of the user-interface of the SortDisplay program is based on Young's application framework [Young 95]. The application framework called MotifApp, supports features common to many Motif applications, including menus, dialogs, multiple top-level windows, etc.

The strategy used here is to make effective use of C++ objects to create higher level user-interface components that combine one or more widgets into a logical grouping. A component not only encapsulates a collection of widgets but defines the behavior of the overall component as well. Nearly all the user-interface components used follow a simple protocol defined by Young [Young 95]. The following features are supported by these components.

- Components create one or more widgets in the class constructor. Normally, callbacks, if any, are registered here as well. Each component creates a single widget, called the base widget that forms the root of a widget tree represented by the class. All other widgets are children or descendants of this base widget.
- Components take a widget as an argument in the constructor. This widget serves as the parent of the component's base widget.
- Components accept a string as an argument in the class constructor. This string is used as the name of the root of the component's widget tree.
- Each component class provides an access method that can be used to retrieve the root widget of the component tree.
- Component classes allow the widget subtree encapsulated by the class to be managed and unmanaged.
- Components handle the widget destruction within the component's widget tree. The widgets encapsulated by an object are destroyed when the object is destroyed.

The basic user-interface component of any Motif application is a widget. The BasicWidget class supports a base widget, which represents the root of every component's widget tree. However, this class itself doesn't create the base widget. The derived classes create the widget of the type needed. BasicWidget class also provides two member functions manage() and unmanage(), which manage or unmanage the component's base widget. These member functions are declared as virtual to allow derived classes to alter this behavior, if needed. This class is intended specifically to serve as a base class for other classes. Its constructor is declared in the protected portion of the class and hence the class cannot be instantiated directly.

The higher level of user-interface components that are derived from the BasicWidget class are shown in Figure 4.

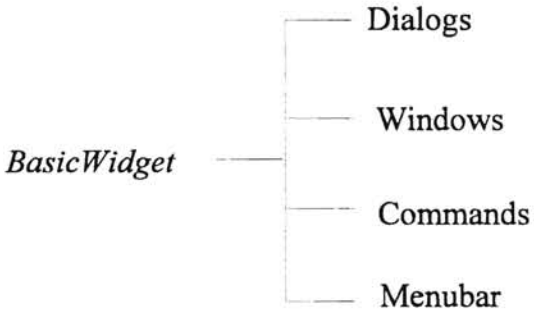

Figure 4. User-Interface Components' Class Hierarchy

The following four sections discuss the higher level classes in detail.

4.4 Dialogs

Dialog windows appear on the screen for a relatively short amount of time and allow

applications to ask questions, display important information, warning messages, and error messages. There are two types of dialog classes, as shown in Figure 5, which are directly derived from the BasicWidget class as implemented here.

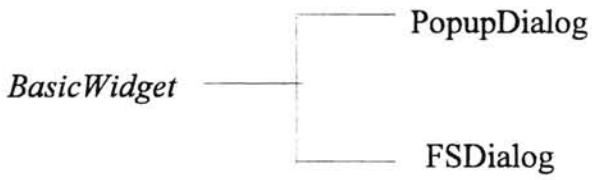

Figure 5. Dialogs' Class Hierarchy

4.4.1 PopupDialog Class

This class is implemented as a dialog manager which creates dialogs based on need, thus increasing speed and efficiency. There are three types of dialogs that are needed by the application: an information dialog, a warning dialog, and an error dialog. The class declares three instances of the PopupDialog, namely theInfoDialog, theWarnDialog, and theErrorDialog which are instantiated at start-up. The class also provides a public member function postMessage() which is used to post messages. This function uses an unused dialog to display the message, if one is available. Otherwise, it creates a new dialog of the type needed to display the message. The dialogs that are created when needed are destroyed immediately after use. The pop up dialogs are shown in Figures 6, 7, and 8.

4.4.2 FSDialog Class

This is a simple class that creates a Motif File Selection Dialog box. The class provides a public member function to return the file name of the file selected by the user. The file selection box is shown in Figure 9.

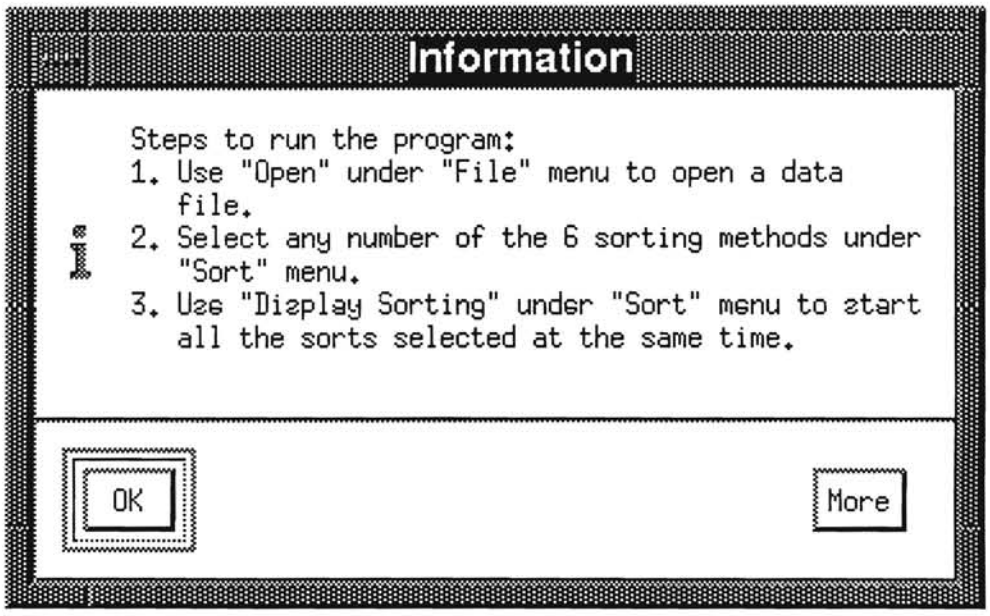

Figure 6. Information Dialog Box

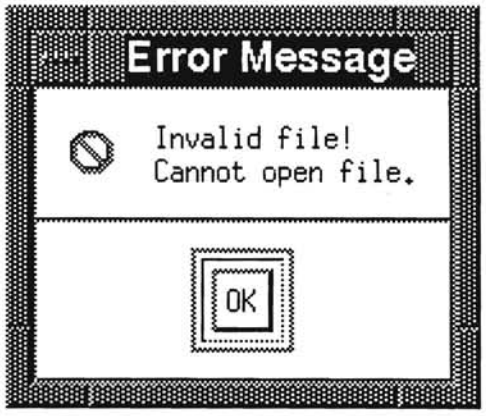

Figure 7. Error Dialog Box

Figure 8. Warning Dialog Box

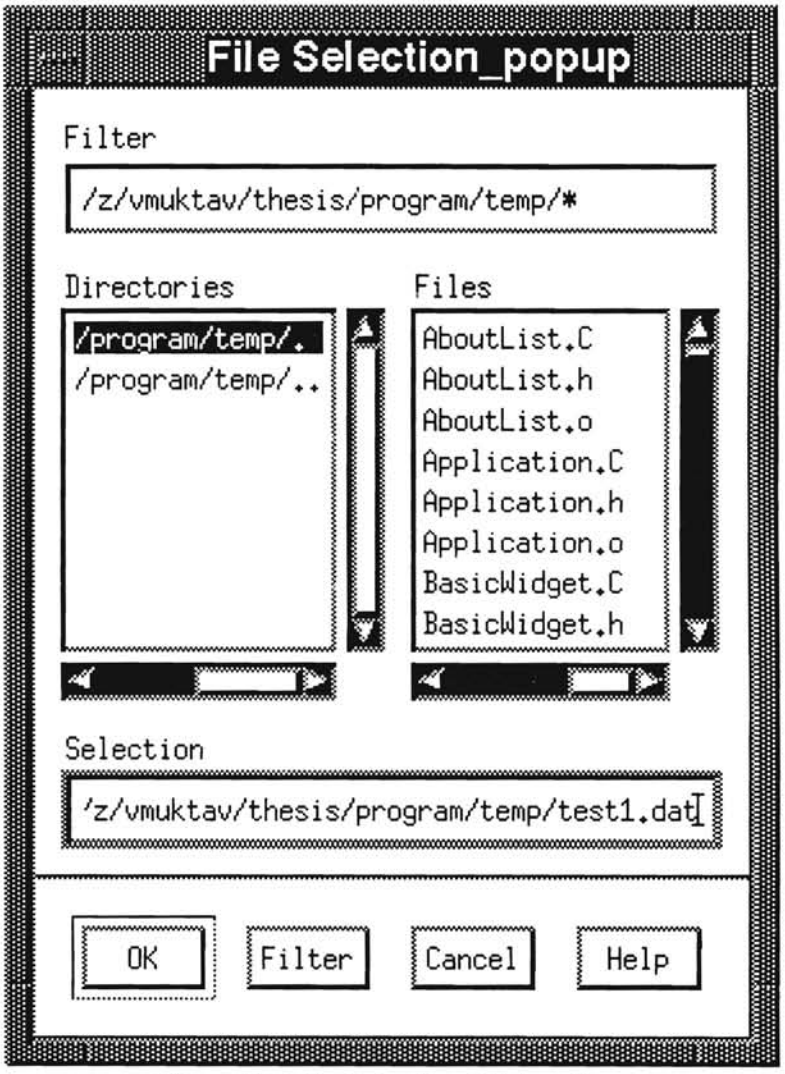

Figure 9. File Selection Dialog Box

4.5 Windows

The SortDisplay program requires multiple windows to be displayed potentially simultaneously including a main window, display windows to display sorting, a speed window to let the user change the speed of display, graph windows to display the complexity analysis graphs of the sorting algorithms, and a statistics window to display the current statistics. All of these windows are part of a class hierarchy as shown in Figure 10.

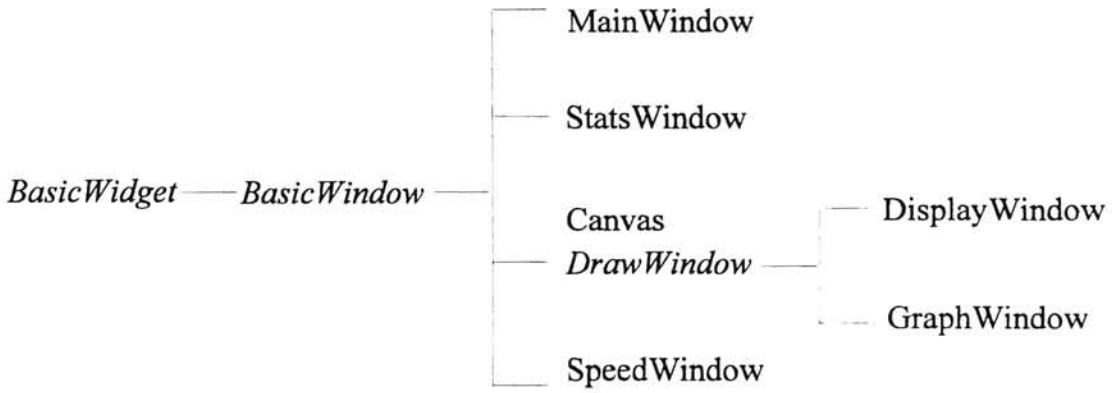

Figure 10. Windows' Class Hierarchy

4.5.1 BasicWindow Class

This class is similar to the BasicWidget class in that it supports a base window like the BasicWidget class which supports a base widget. This class is derived from BasicWidget and, in addition to the base widget of the BasicWidget, supports a base window. This class itself doesn't create the widget or the window but provides two absolute virtual member functions createWidget() and createShell(), which are implemented by derived classes to create the base widget and the base window of the type and dimensions needed. The BasicWindow class also provides two public member functions, manage() and unmanage()

for popping-up and popping-down the window. These member functions are declared virtual so that they can be modified, if needed.

4.5.2 DrawWindow Class

As discussed earlier in the chapter (Subsection 4.2.2), the Canvas class provides the pixmap and the drawing functions that can be used to draw on the pixmap. However, a pixmap doesn't appear on the screen directly. The contents of the pixmap have to be copied to a window. The DrawWindow class provides the window and the drawing area to display the drawing on the pixmap on the screen. This is an abstract class which creates a window and a DrawingArea widget to which the pixmap can be copied. The class also supports a virtual member function (to handle expose-callback events) and is implemented by the derived classes to actually copy the contents of the pixmap onto the window.

The DisplayWindow class is derived from the DrawWindow class to implement dynamic display of sorting. A snap shot of a display window while sorting is shown in Figure 11.

GraphWindow class is derived from the DrawWindow class to implement the complexity analysis graphs of the sorting algorithms. Figures 12a and 12b show pictures of two types of graphs implemented.

4.5.3 MainWindow Class

The main window of an application is the most visible and most used of all the windows in an application. It is the focal point of the user's interactions with the program,

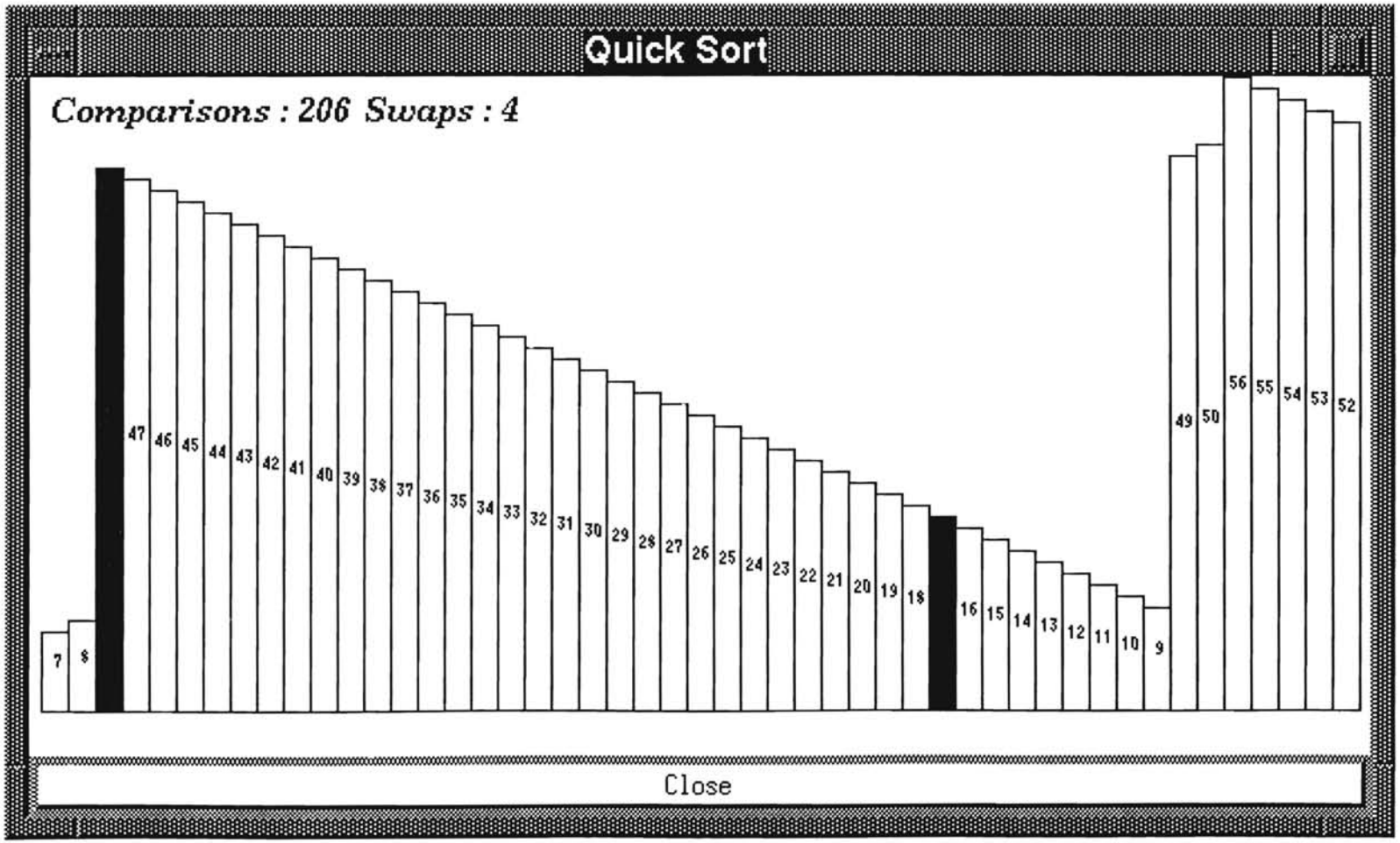

Figure 11. Display Window

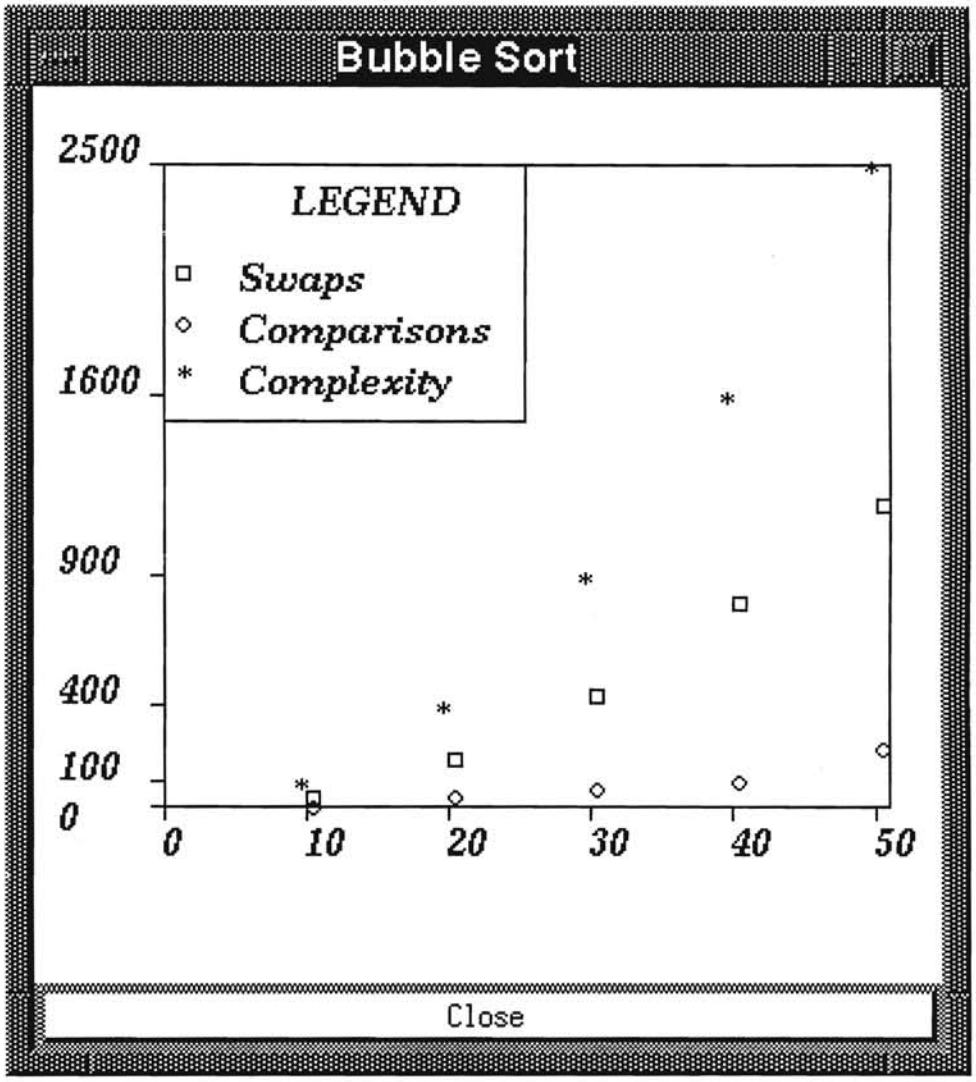

Figure 12a. Graph Window

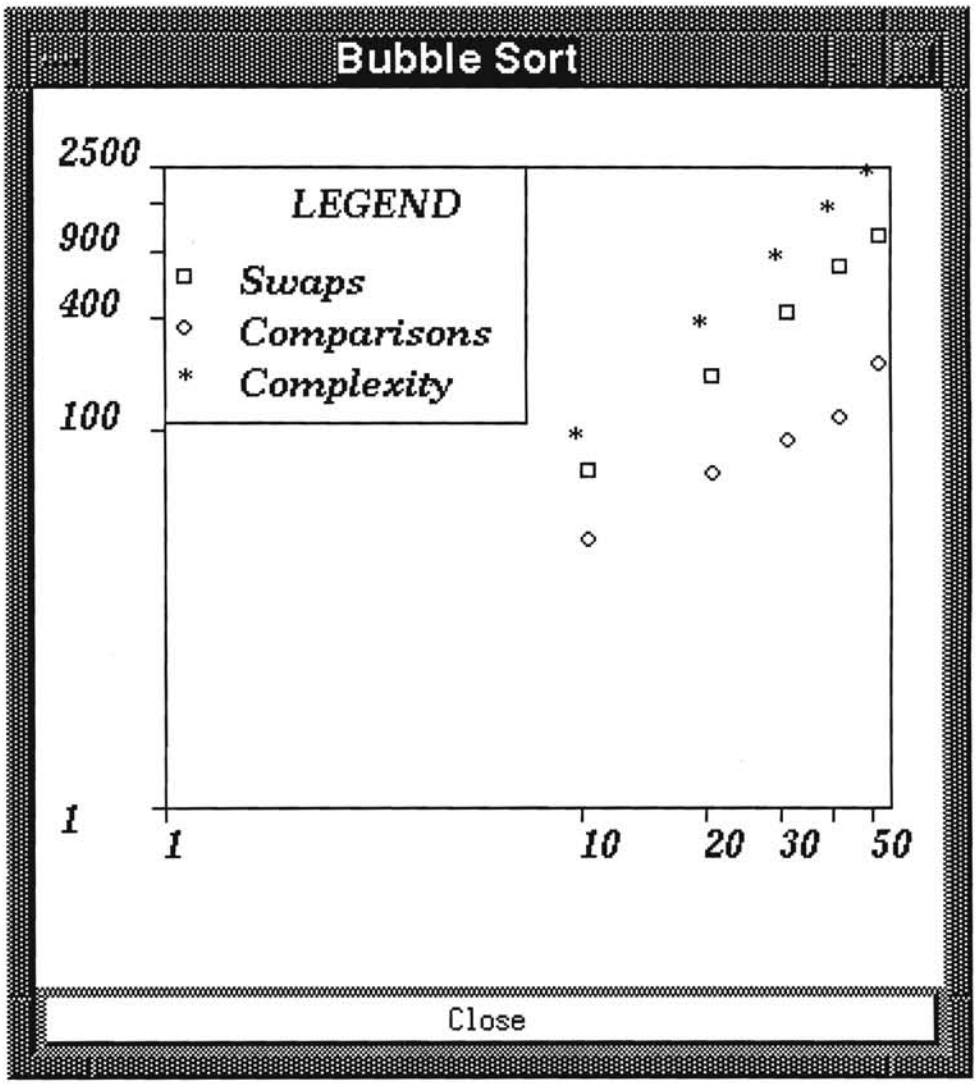

Figure 12b. GraphWindow

and it is typically the place where the application provides most of its visual feedback [Heller 91].

The MainWindow Class creates the main window of the SortDisplay program. The main window also creates the initial screen of the application which will be discussed later in the chapter. This class also manages the layout of the menubar. Figure 13 shows the main window including the menubar.

4.5.4 Other Windows

The speed window created by the SpeedWindow class gives the user the option to change the speed of sorting display. The speed window is shown in Figure 14.

The statistics window is created by the StatsWindow class and shows the current statistics of sorting, as seen in Figure 15.

4.6 Commands

Nearly every user action in an interactive application can be thought of as a command [Young 95]. A command can be modeled as an object with the following responsibilities.

- Initiate an action.
- Activate and deactivate.
- Maintain lists of dependent objects.
- Activate and deactivate dependent commands when executed.

The commands implemented here are all part of a commands' class hierarchy as shown in Figure 16.

4.6.1 CmdInterface Class

It is common for a command object to represent an action initiated by the user

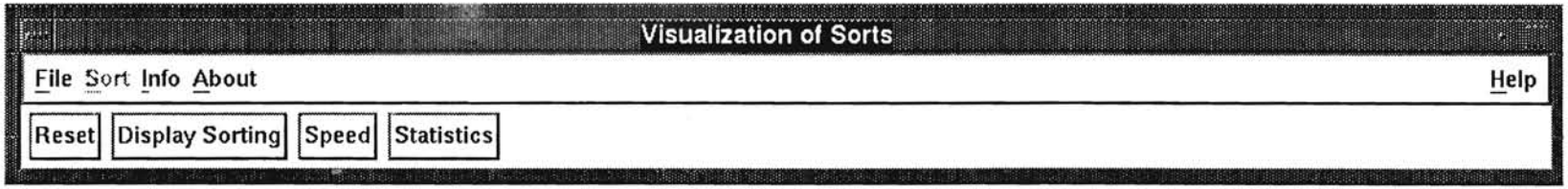

Figure 13. Main Window with Menubar

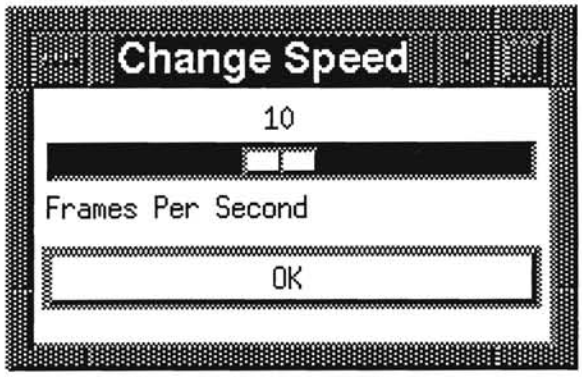

Figure 14. Speed Window

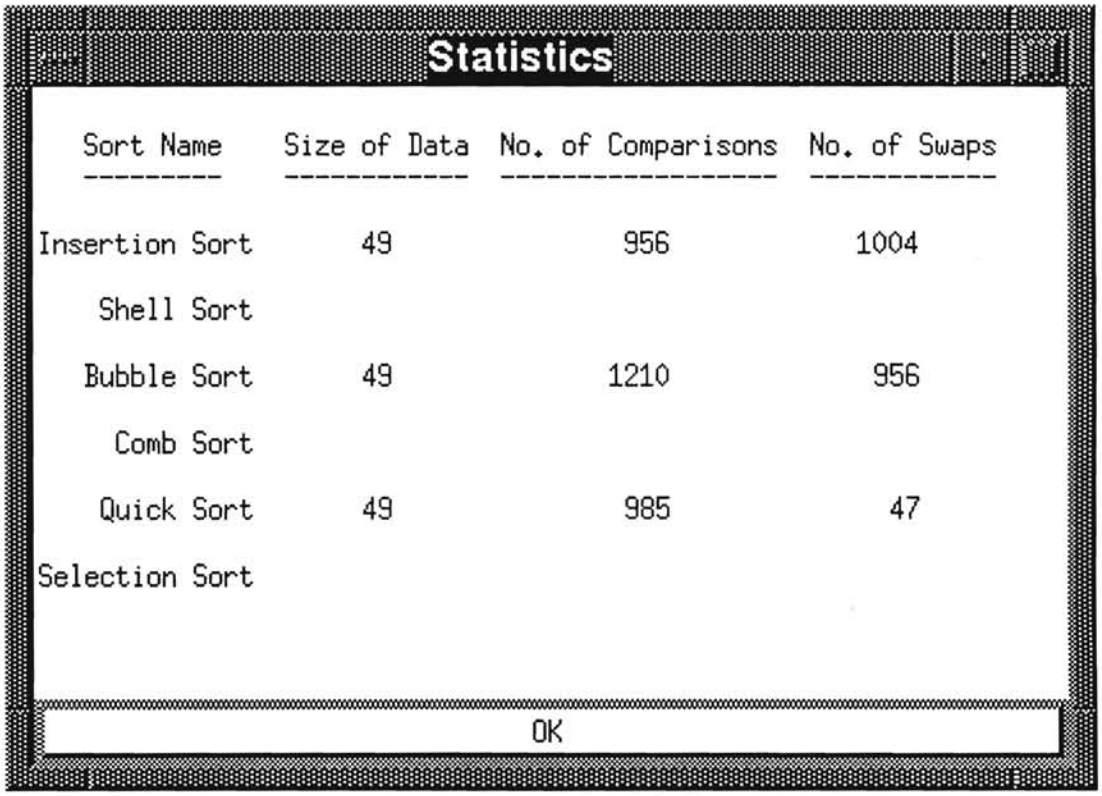

Figure 15. Statistics Window

through some user-interface components such as a button widget or an entry in a menu. The CmdInterface class is an abstract class derived from BasicWidget. The class doesn't create any widgets itself but supports a callback function, executeCallback() that can be used to execute the command. However, since the CmdInterface class doesn't create a widget, it cannot register the callback, but derived classes can register this callback function with the appropriate widget. The CmdInterface class also supports two virtual member functions activate() and deactivate(). These functions activate and deactivate the widget or widgets supported by a CmdInterface object. The CmdInterface class has no public protocol and the entire protocol is protected and is accessible only to derived classes.

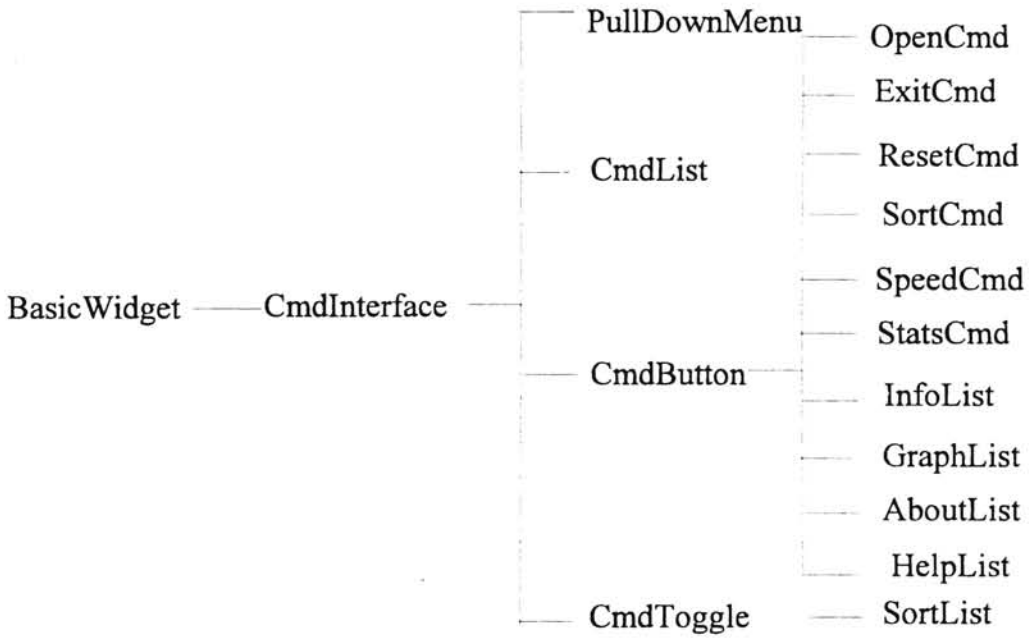

Figure 16. Commands' Class Hierarchy

4.6.2 CmdList Class

The derived classes of CmdInterface often need to deal with lists of CmdInterface objects. It is useful to create a simple list that makes it easier to deal with these lists of

objects. Its public protocol has functions activate() and deactivate() which activate and deactivate the list of CmdInterface objects the class maintains.

4.6.3 CmdButton Class

The CmdButton class (derived from CmdInterface) provides a push button as the user-interface component, and registers the callback function as an XmNactviateCallback function to be called when the user pushes the push button. The CmdButton class defines only a constructor. All the other functions are inherited from CmdInterface. CmdButton is still an abstract class since the executeCallback() function is not yet implemented by this class. This function defines the actual action to be taken and is defined by the derived classes.

4.6.4 CmdToggle Class

The CmdToggle class (derived from CmdInterface) provides a toggle button as the user-interface object and registers the callback function as an XmNactivateCallback function to be called when the user toggles the toggle button.

4.6.5 PullDownMenu Class

This class is designed to put together a set of commands and create a pull-down cascading menu. This class has a public function add(), which takes CmdInterface objects and creates cascading menus from them.

4.7 Menubar

Menus provide the user with a set of choices in an application without complicating its normal visual appearance [Heller 91]. Menubar provides a good place for putting

together pop-up menus.

The MenuBar class encapsulates the process of constructing a Motif menubar with multiple pull down menu panes. The MenuBar class supports a public member function add(), which takes CmdInterface objects and creates a menubar from these objects. The objects that are passed to the add() function are PullDownMenu objects so that the menubar has a set of pull-down menus.

4.8 Putting It All Together

The previous sections discussed how the interface objects, windows used for display, and dialogs have been designed and implemented. Now that the independent objects have been created, the next task is to put them all together and set up inter-dependencies and controls so that all the disparate elements work as part of one application to constitute the tool.

4.8.1 ControlPanel Class

This class creates all the instances of the command classes that provide the user-interface components with which the user interacts. The class then organizes these user-interface objects into logical pull-down menus that are then attached to a menubar. The class also instantiates the dialogs that are declared externally by the PopuDialog class.

The class supports a public member function setControls(), which sets up inter-dependencies among various classes. The function then attaches the menubar to the main window so that it is included in the main window layout. And finally, it manages the main

window so that the application starts up with the main window showing the initial screen (as shown in Figure 17).

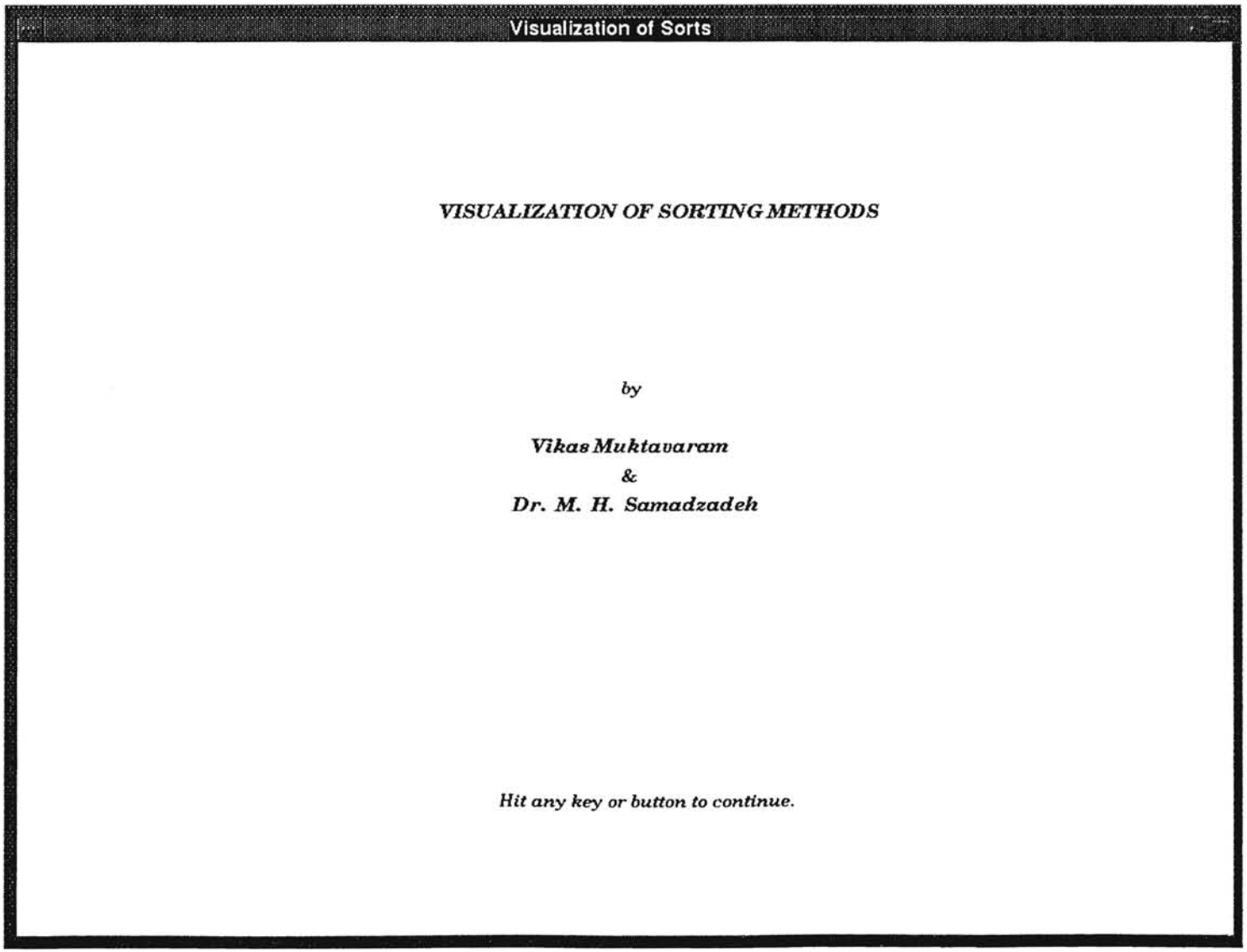

Figure 17. Initial Screen

CHAPTER V

TESTING AND EVALUATION

5.1 User Appraisal

The testing phase of the tool, SortDisplay, developed as part of the thesis, was very exciting for the author for at least two reasons: one being that after almost six months of development, the tool finally got to see the daylight; and the other is that the very idea of having the tool evaluated by other users was exciting in itself. An early version of the tool was evaluated by a variety of users of the Computer Science Department's Sequent Symmetry S/81 machine including faculty and staff members as well as a number of former and current graduate students.

Based on the recommendations, comments, and suggestions of the users, a variety of modifications, improvements, and refinements came over the tool. The following changes were effected.

1. A "Help" menu and an "About" menu were added. The "Help" menu gives information about input data, sorting, display, and other options the tool provides. The "About" menu provides general information on the program and on using the tool.

2. A major bug was found in the program thanks to Mr. Mark Vasoll. The program had a memory leak and was crashing when running on a PC using the XVision™ software. At the

suggestions of Mr. Mark Vasoll and Dr. Mansur Samadzadeh, it was determined that the program was using an enormous amount of memory upon unmanaging the Pop-up dialogs. The reason for the memory leak was that the Pop-up dialogs were being unmanaged twice: once because the dialogs had the Motif built-in "auto-unmanage" option on, and then again by the program. The problem was fixed by not having the program unmanage a dialog that had already been unmanaged.

3. It was pointed out by many users that while sorting, swapping was not shown prominently because rectangles that are being swapped or compared were completely shaded. To distinguish a swap operation from a comparison operation, rectangles are now partially shaded when numbers are swapped and are still fully shaded when compared.

4. It was felt that when all the sorting display windows were simultaneously opened. the response to the user input was extremely slow. The reason for the slow response is explained here. Display involves drawing a new scene for every interval of time and displaying it on the next frame. To draw, a graphics context is needed and the graphics context was being changed every time a new frame was drawn. And every time the graphics context is changed, the information about the change has to travel over the network to the server, which is time consuming. This problem was fixed by creating all the graphics contexts needed at program initialization so that the graphics context doesn't need to be changed often while the program is running. This certainly did speed up the response time.

5. Some of the other minor changes include: a) synchronizing the closing and opening of the display windows with the selecting or unselecting of sorts in the Sort menu, b) displaying user-friendly error and warning messages, c) using bigger and bolder fonts for the menu

item names, d) displaying the values represented by rectangles while sorting.

CHAPTER VI

SUMMARY AND FUTURE WORK

The usefulness of a graphical tool for visualization of sorting methods was discussed in Chapter I. Chapter II presented a description of the concept of visualization and various sorting methods. Chapter III discussed the implementation platform and environment for the SortDisplay tool. Chapter IV talked about the design and implementation issues of the tool. Chapter V described the changes done to the initial version of the tool based on comments from a number of users of the tool.

Although sorting could be done on any well-ordered set including integers, real numbers, structures, or programs, for the sake of convenience and ease of representation and visualization, only the sorting of nonnegative integers was considered. This thesis involved developing a graphical tool for the visualization of six sorting algorithms.

The SortDisplay tool was designed to be an educational tool running on the Computer Science Department's Sequent S/81. Such a tool would help users understand the various sorting algorithms.

Future versions of the tool could include the following modifications.

1. If the system can support sound, a narrative attached to the dynamic display would make the display more interesting and helpful.
2. The graphs of the sorting algorithms could be dynamic based on the sorting information obtained from each new run.

3. Other graphs such as "nlogn vs n^2" or "n^2 vs n^2" could be implemented.

4. New sorting methods could be incorporated into the tool.

5. Instead of sorting only integers as the tool does now, sorting of names, etc. based on lexicographical or canonical ordering could be included.

REFERENCES

[Aho et al. 83] Alfred V. Aho, John E. Hopcroft, and Jeffrey Ullman, *Data Structures and Algorithms*, Addison - Wesley Publishing Company, Reading, MA, 1983

[Barkakati 91] Nabajyoti Barkakati, *X Window System™ Programming*, Macmillan Computer Publishing, Carmel, IN, 1991.

[Bernstein 95] Daniel Bernstein, *Using Motif with C++*, SIGS Books, New York, NY, 1995.

[Box and Lacey 91] Richard Box and Stephen Lacey, "A Fast, Easy Sort", *Byte*, Vol. 16, No. 4, pp. 315 - 320, April 1991.

[Dabbi 94] Jaganath Dabbi, *Towards a Graphical Queuing Network Tool*, Master of Science Thesis, Computer Science Department, Oklahoma State University, Stillwater, OK, 1994.

[Du 93] Haibo Du, *Display of Sequent Symmetry S/81 Performance Using X Window System Facilities*, Master of Science Thesis, Computer Science Department, Oklahoma State University, Stillwater, OK, 1993.

[Flores 69] Ivan Flores, *Computer Sorting*, Prentice-Hall, Englewood Cliffs, NJ, 1969.

[Gerald and Ueberhuber 94] Tomas Gerald and Christoph W. Ueberhuber, *Visualization of Scientific Parallel Programs*, Springer-Verlag, Berlin, Germany, 1994.

[Gershon 92] Nahum D. Gershon, "From Perception to Visualization", *Computer Graphics*, Vol. 26, No. 2, pp. 414 - 417, July 1992.

[Heller 91] Dan Heller, *Motif Programming Manual*, O'Reilly and Associates, Inc., Sebastapol, CA, 1991.

[House 82] William C. House, *Interactive Computer Graphics Systems*, Petrocelli Books, Inc., New York, NY, 1982.

[Knuth 73] Donald E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and*

Searching, Addison-Wesley Publishing Company, Inc., Reading, MA, 1973.

[Koshy 95] Bobby S. Koshy, *Towards a Graphical Deadlock Analysis Tool*, Master of Science Thesis, Computer Science Department, Oklahoma State University, Stillwater, OK, 1995.

[Lee 88] Wilson Lee, *An Implementation of a Data Structure Display System*, Master of Science Thesis, Computing and Information Science Department, Oklahoma State University, Stillwater, OK, 1988.

[Manber 89] Udi Manber, *Introduction to Algorithms: A Creative Approach*, Addison-Wesley Publishing Company, Inc., Reading MA, 1989.

[Mansfield 91] Niall Mansfield, *The X Window System: A User's Guide*, Addison-Wesley Publishing Company, Reading, MA, 1991.

[Nye 90] Andrian Nye, *X Protocol Reference Manual for Version 11*, O'Reilly & Associates, Inc., Sebastapol, CA, 1990.

[Ousterhout 94] John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley Publishing Company, Reading, MA, 1994.

[Rich 72] Robert P. Rich, *Internal Sorting Methods Illustrated with PL/1 Programs*, Prentice-Hall, EngleWood Cliffs, NJ, 1972.

[Rost 90] Randi J. Rost, *X and Motif Quick Reference Guide*, Digital Press, X and Motif Series, Bedford, MA, 1990.

[Sedgewick 90] Robert Sedgewick, *Algorithms in C*, Addison-Wesley Publishing Company, Inc., Reading, MA, 1990.

[Shell 59] Donald L. Shell, "A High-Speed Sorting Procedure", *Communications of the ACM*, Vol. 2, No. 7, pp. 30-32, July 1959.

[Shneiderman 87] Ben Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley Publishing Company, Reading, MA, 1987.

[Smith 91] Jerry D. Smith, *Object Oriented Programming with the X Window System Toolkits*, John Wiley & Sons, Inc., New York, NY, 1991.

[Sorting out Sorting 81] *Sorting out Sorting*, The Dynamic Graphics Project Computer Systems Research Group, University of Toronto, Toronto, Canada, 1981.

[Su 93] Yuh-Ching Su, *An Empirical Study of Combsort and Ways to Improve It*, Master of Science Thesis, Computer Science Department, Oklahoma State University, Stillwater, OK, 1993.

[Weiss 91] Mark A. Weiss, "Empirical study of the expected running time of Shellsort", *The Computer Journal*, Vol. 34, No. 1, pp. 88 - 91, February 1991.

[Weiss 93] Mark A. Weiss, *Data Structures and Algorithm Analysis in C*, Benjamin/Cummins Publishing Company, Inc., Redwood City, CA, 1993.

[Young 95] Douglas A. Young, *Object-Oriented Programming with C++ and OSF/Motif*, Prentice Hall, Upper Saddle River, NJ, 1995.

APPENDICES

APPENDIX A

GLOSSARY

Client - Server Model:	A server process in a client - server model provides some services to the other processes. These other processes are known as clients. In the X Window System, the server controls all input and output devices. An application is a client process that utilizes the services provided by the server.
Graphical User Interface:	A visual representation of some of the functionality of a computer that can be manipulated in a friendly, easy to use, and non-programmatic manner.
GUI:	Graphical User Interface.
Sorting:	The ordering or rearrangement of items into ascending or descending order.
Tcl:	Tcl stands for Tool Command Language. It is a scripting language that is used for developing and using graphical user interface applications.
Tk:	A toolkit based on Tcl that helps users create graphical user interfaces for the X11 Window System by writing Tcl scripts.
Visualization:	The process of transforming information into a visual form.
Widget:	A user interface mechanism comprising data structures and the associated procedures that can be displayed in different ways such as menus, dialog boxes, or windows.
Window:	A rectangular area on the screen that belongs to a particular application.

Window Manager:	The program that manages the display of windows and their manipulation on the screen.
X:	A networked, portable, and transparent windowing system.
X Toolkit Instrinsics:	A library of functions, procedures, and data structures built on top of Xlib that makes application programming much easier compared to working with Xlib functions.
X Window System:	A network transparent and hardware independent base layer that provides services to the Graphical User Interfaces.

APPENDIX B

TRADEMARK INFORMATION

DEC:	A registered trademark of Digital Equipment Corporation.
DYNIX/ptx:	A registered trademark of Sequent Computer Systems, Inc.
Motif and OSF:	Registered trademarks of the Open Software Foundation.
Sequent S/81:	A registered trademark of Sequent Computer Systems, Inc.
UNIX:	A registered trademark of AT&T.
XVision:	A trademark of Visionware Ltd.
X Window System:	A registered trademark of the Massachusetts Institute of Technology.

APPENDIX C

PROGRAMMER'S GUIDE

1. Description

The tool developed, called SortDisplay, is a graphical tool for the dynamic visualization of sorting methods. The program was coded in the C++ programming language using the Motif toolkit, Xt Intrinsics, and Xlib. The program has 40 classes and each class is declared in a file class.h (class is the name of the class) and implemented in class.C. The tool displays sorting, provides information on the sorting algorithms, and gives statistics on the current run.

2. Implementation

Sorting methods are implemented as derived classes of the abstract class "Sort." Any new sorting method can be added to the program by implementing the method as a derived class of "Sort." The "SortInfo" class maintains the sorting information, i.e., the number of comparisons and swaps, at each stage of the sorting process.

To add any new information on the sorting methods, a new text can be added to the file "InfoList.C", and the help information can be updated in the file "HelpList.C". The information needed to draw a graph for a sorting algorithm is the empirical statistics information (number of comparisons and swaps) and the average case complexity of the algorithm. The "GraphList" class maintains the information needed to draw graphs.

The maximum size of input data is restricted to 50. The "OpenCmd" class reads only the first 50 elements of the input data file. While reading the input file, any number greater than 99 is truncated to 99. These restrictions can be changed by modifying the criteria in "OpenCmd.C".

APPENDIX D

PROGRAM LISTING

```
////////////////////////////////////////////////////////////////////////////////
//                              Program : SortDisplay
//                               Author : Vikas Muktavaram
//                                 Date : 30 June 1996
//                  Programming Language : C++
//                  Toolkits & Libraries : Motif, Xt Intrinsics, and Xlib
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// Main.C : Generic main() routine.
//          Instantiate global Application class and ControlPanel class objects.
//          Register controls, initialize the application and handle events.
////////////////////////////////////////////////////////////////////////////////

#include "Application.h"
#include "ControlPanel.h"

void main(unsigned argc,char** argv)
{
  // Instantiate global objects.

  Application*  theApplication = new Application(&argc,argv);
  ControlPanel* theControls    = new ControlPanel();

  // Register controls and initalize.

  theApplication->registerControl(theControls);
  theApplication->initialize();

  // Enter event loop.

  theApplication->handleEvents();
}
////////////////////////////////////////////////////////////////////////////////
// Application.h
////////////////////////////////////////////////////////////////////////////////

#ifndef APPLICATION_H
#define APPLICATION_H

#include <Xm/Xm.h>

// Forward Declaration.

class ControlPanel;

class Application
{
  // Declare friends

  friend void main(unsigned,char**);
  friend class ControlPanel;

  private:

    Widget         _toplevel;   // Top level shell widget.
    ControlPanel*  _controls;   // Application controls.
```

```
    // Support commonly needed data structures as a convenience.

    Display*     _display;
    XtAppContext _appContext;

    void registerControl(ControlPanel*);  // Register controls.

    // Functions to handle Xt interface.

    void initialize(void);
    void handleEvents(void);

  public:

    Application(unsigned*,char**);
    ~Application(void);

    Widget       toplevel(void);  // Function returns toplevel widget.

    // Functions to return the common data structures needed by the application.

    Display*     display(void);
    XtAppContext appContext(void);
};

// Pointer to single global instance.

extern Application* theApplication;

#endif
////////////////////////////////////////////////////////////////////////////
// Application.C
////////////////////////////////////////////////////////////////////////////

#include "Application.h"
#include "ControlPanel.h"

#include <stdlib.h>
#include <iostream.h>
#include <assert.h>

Application* theApplication = NULL;

Application::Application(unsigned* argc,char** argv)
{
  char* appName = argv[0];

  // Set the global Application pointer.

  theApplication = this;

  // Initialize the toolkit.

  XtToolkitInitialize();

  // Create and save the application context.

  _appContext = XtCreateApplicationContext();

  // Create and save a pointer to the X display structure

  _display    = XtOpenDisplay(_appContext,
                              NULL,
                              NULL,
                              appName,
                              NULL,
                              0,
                              argc,
                              argv);


  // ERROR!

  if(_display == NULL)
  {
    cout << "Sorry! Cannot open display." << endl;
    exit(0);
```

```
  }

  // Create the top shell.

  _toplevel = XtVaAppCreateShell(NULL,
                                 appName,
                                 applicationShellWidgetClass,
                                 _display,
                                 NULL);

  // Make sure the shell isn't visible.

  XtVaSetValues(_toplevel,
                XmNmappedWhenManaged,FALSE,
                XmNx, DisplayWidth(_display,0)/2,
                XmNy, DisplayWidth(_display,0)/2,
                XmNwidth, 1,
                XmNheignt,1,
                NULL);
}

Application::~Application(void)
{
  if(_toplevel)
  {
    XtDestroyWidget(_toplevel);
  }

  if(_appContext)
  {
    XtDestroyApplicationContext(_appContext);
  }
}

void Application::registerControl(ControlPanel* controls)
{
  _controls = controls;
}

Widget Application::toplevel(void)
{
  return _toplevel;
}

Display* Application::display(void)
{
  return _display;
}

XtAppContext Application::appContext(void)
{
  return _appContext;
}

void Application::initialize(void)
{
  // Initialize and manage the application window

  assert(_controls != NULL);
  _controls->control();

  // Force the shell to exist

  XtRealizeWidget(_toplevel);
}

void Application::handleEvents(void)
{
  // Just loop forever

  XtAppMainLoop(_appContext);
}
////////////////////////////////////////////////////////////////////////////
// ControlPanel.h : Create all the user-interface objects needed and set
//                  controls.
////////////////////////////////////////////////////////////////////////////
```

```
#ifndef CONTROLPANEL_H
#define CONTROLPANEL_H

#include "SortCmd.h"
#include "ResetCmd.h"
#include "SortList.h"
#include "HelpList.h"
#include "AboutList.h"

// Forward declarations.

class MenuBar;
class MainWindow;
class PullDownMenu;
class AboutCmd;
class OpenCmd;
class ExitCmd;
class SpeedCmd;
class StatsCmd;
class GraphList;
class InfoList;

class ControlPanel
{
  private:

    // Menubar.

    MenuBar* _menuBar;

    // Main window.

    MainWindow* _mainWindow;

    // Cascade menus.

    PullDownMenu* _fileMenu;
    PullDownMenu* _sortMenu;
    PullDownMenu* _infoMenu;
    PullDownMenu* _helpMenu;
    PullDownMenu* _aboutMenu;

    // File menu items.

    OpenCmd* _open;
    ExitCmd* _exit;

    // Sort menu items.

    SortList* _sort[NoOfSorts];
    SortCmd*  _sortCmd;
    ResetCmd* _resetCmd;

    // Options menu items.

    SpeedCmd* _speedCmd;

    // Info menu items.

    StatsCmd*     _statsCmd;
    PullDownMenu* _graphSubMenu;
    PullDownMenu* _infoSubMenu;

    // Graph submenu items.

    GraphList* _graph[NoOfSorts];

    // Info submenu items.

    InfoList* _info[NoOfSorts];

    // Help menu items.

    HelpList* _help[NoOfHelps];

    // About menu items.
```

```
    AboutList* _about[NoOfAbouts];

  public:

    ControlPanel(void);
    ~ControlPanel(void);

    void control(void);
};

#endif
////////////////////////////////////////////////////////////////////////////
// ControlPanel.C : Create all the user-interface objects needed and set
//                  controls.
////////////////////////////////////////////////////////////////////////////

#include "ControlPanel.h"
#include "MenuBar.h"
#include "MainWindow.h"
#include "PullDownMenu.h"
#include "AboutList.h"
#include "OpenCmd.h"
#include "ExitCmd.h"
#include "SpeedCmd.h"
#include "StatsCmd.h"
#include "GraphList.h"
#include "InfoList.h"
#include "PopupDialog.h"

// Constructor instantiates all the user-interface objects needed.

ControlPanel::ControlPanel(void)
{
  // Initialize the menubar.

  _mainWindow = new MainWindow("Visualization of Sorts");
  _mainWindow->initialize();
  _mainWindow->manage();

  // Create a menubar.

  _menuBar    = new MenuBar("MenuBar",_mainWindow->baseWidget());

  // Create the menu items.

  _fileMenu  = new PullDownMenu("File",_menuBar->baseWidget(),TRUE);
  _sortMenu  = new PullDownMenu("Sort",_menuBar->baseWidget(),FALSE);
  _infoMenu  = new PullDownMenu("Info",_menuBar->baseWidget(),TRUE);
  _aboutMenu = new PullDownMenu("About",_menuBar->baseWidget(),TRUE);
  _helpMenu  = new PullDownMenu("Help",_menuBar->baseWidget(),TRUE,TRUE);

  // Create sub-menu items in File menu.

  _open = new OpenCmd(_fileMenu->baseWidget(),TRUE);
  _exit = new ExitCmd(_fileMenu->baseWidget(),TRUE);

  // Create sub-menu items in Info menu.

  _graphSubMenu = new PullDownMenu("Graph",_infoMenu->baseWidget(),TRUE);
  _infoSubMenu  = new PullDownMenu("Info",_infoMenu->baseWidget(),TRUE);

  for(int i = 0;i < NoOfSorts;i++)
  {
    _sort[i]  = new SortList(sortName[i],sortClass[i],
                                          _sortMenu->baseWidget(),TRUE);
    _graph[i] = new GraphList(sortName[i],sortClass[i],
                                       _graphSubMenu->baseWidget(),TRUE);
    _info[i]  = new InfoList(sortName[i],sortClass[i],
                                        _infoSubMenu->baseWidget(),TRUE);
  }

  // Create Help menu.

  for(i = 0;i < NoOfHelps;i++)
  {
    _help[i] = new HelpList(helpName[i],helpClass[i],
                                          _helpMenu->baseWidget(),TRUE);
```

```
  }

  // Create About menu.

  for(i = 0;i < NoOfAbouts;i++)
  {
    _about[i] = new AboutList(aboutName[i],aboutClass[i],
                                                    _aboutMenu->baseWidget(),TRUE);
  }

  // Command buttons that appear on the main window.

  _resetCmd = new ResetCmd(_mainWindow->workArea(),TRUE);
  _sortCmd  = new SortCmd(_mainWindow->workArea(),TRUE);
  _speedCmd = new SpeedCmd(_mainWindow->workArea(),TRUE);
  _statsCmd = new StatsCmd(_mainWindow->workArea(),TRUE);

  // Instantiate the global dialog objects.

  theInfoDialog  = new PopupDialog("Information",INFO);
  theErrorDialog = new PopupDialog("Error Message",ERROR);
  theWarnDialog  = new PopupDialog("Warning Message",WARN);
}

// Free all the objects create in the constructor.

ControlPanel::~ControlPanel(void)
{
  delete _menuBar;
  delete _mainWindow;

  delete _fileMenu;
  delete _sortMenu;
  delete _infoMenu;
  delete _aboutMenu;
  delete _helpMenu;

  delete _open;
  delete _exit;

  for(int i = 0;i < NoOfSorts;i++)
  {
    delete _sort[i];
    delete _graph[i];
    delete _info[i];
  }

  delete _graphSubMenu;
  delete _infoSubMenu;

  for(i = 0;i < NoOfHelps;i++)
  {
    delete _help[i];
  }

  for(i = 0;i < NoOfAbouts;i++)
  {
    delete _about[i];
  }

  delete _sortCmd;
  delete _resetCmd;
  delete _speedCmd;
  delete _statsCmd;

  delete theInfoDialog;
  delete theErrorDialog;
  delete theWarnDialog;
}

// Set up interdependencies.

void ControlPanel::control(void)
{
  _mainWindow->addMenuBar(_menuBar);

  _menuBar->add(_fileMenu);
```

```
  _fileMenu->add(_open);
  _fileMenu->add(_exit);

  _open->addToActivationList(_sortMenu);

  _menuBar->add(_sortMenu);

  for(int i = 0;i < NoOfSorts;i++)
  {
    _sortMenu->add(_sort[i]);
    _sort[i]->setControls(_open,_speedCmd);
  }

  _sortCmd->add(_sort);
  _resetCmd->add(_sort);

  _menuBar->add(_aboutMenu);
  _menuBar->add(_infoMenu);

  _infoMenu->add(_infoSubMenu);
  _infoMenu->add(_graphSubMenu);

  _statsCmd->add(_sort);

  for(i = 0;i < NoOfSorts;i++)
  {
    _infoSubMenu->add(_info[i]);
    _graphSubMenu->add(_graph[i]);
  }

  _menuBar->add(_helpMenu);

  for(i = 0;i < NoOfHelps;i++)
  {
    _helpMenu->add(_help[i]);
  }

  for(i = 0;i < NoOfAbouts;i++)
  {
    _aboutMenu->add(_about[i]);
  }

  _mainWindow->add(_resetCmd);
  _mainWindow->add(_sortCmd);
  _mainWindow->add(_speedCmd);
  _mainWindow->add(_statsCmd);
}
////////////////////////////////////////////////////////////////////////////
// BasicWidget.h : Abstract class to support a base widget.
////////////////////////////////////////////////////////////////////////////

#ifndef BASICWIDGET_H
#define BASICWIDGET_H

#include <Xm/Xm.h>

class BasicWidget
{
  protected:

    char*  _widgetName;           // Base widget name.
    Widget _widget;               // Base widget.

    // Declare constructor as a protected member function to prevent
    // instantiation.

    BasicWidget(const char*);     // Constructor requires only a name.

  public:

    virtual ~BasicWidget(void);

    // Manage and unmanage the widget tree.

    virtual void manage(void);
    virtual void unmanage(void);
```

```
    const Widget baseWidget(void);
};

#endif
////////////////////////////////////////////////////////////////////////////
// BasicWidget.C : Abstract class to support a base widget.
////////////////////////////////////////////////////////////////////////////

#include "Application.h"
#include "BasicWidget.h"

#include <stdlib.h>
#include <assert.h>

BasicWidget::BasicWidget(const char* name)
{
  _widget = NULL;                  // This class doesn't create the base widget.

  assert(name != NULL);
  _widgetName = XtNewString(name);
}

BasicWidget::~BasicWidget(void)
{
  // Destroy the wideget if it still exists.

  if(_widget)
  {
    XtDestroyWidget(_widget);
  }

  XtFree(_widgetName);
}

const Widget BasicWidget::baseWidget(void)
{
  return _widget;
}

void BasicWidget::manage(void)
{
  assert(_widget);
  XtManageChild(_widget);
}

void BasicWidget::unmanage(void)
{
  assert(_widget);
  XtUnmanageChild(_widget);
}
////////////////////////////////////////////////////////////////////////////
// BasicWindow : Abstract class to support a base window.
////////////////////////////////////////////////////////////////////////////

#ifndef BASICWINDOW_H
#define BASICWINDOW_H

#include "BasicWidget.h"

class BasicWindow:public BasicWidget
{
  protected:

    char*  _windowName;            // Base window name.
    Widget _window;                // Base window.

  public:

    // Constructor requires names for the window and the widget

    BasicWindow(const char*,const char*);

    virtual ~BasicWindow(void);

    // Functions to rename and resize the base window.

    void rename(char*);
```

```
    void resize(int,int);

    // Manage and unmanage the window tree.

    virtual void manage(void);
    virtual void unmanage(void);

    virtual void iconify(void);

    const Widget baseWindow(void);

    // Create the base window.

    virtual void initialize(void);

    // Create a shell and a widget to create the window. Absolute virtual
    // member functions to be implemented in the derived classes.

    virtual void createShell(void)  = 0;
    virtual void createWidget(void) = 0;
};

#endif
////////////////////////////////////////////////////////////////////////////
// BasicWindow.C : Abstract class to support a base window.
////////////////////////////////////////////////////////////////////////////

#include "Application.h"
#include "BasicWindow.h"

#include <stdlib.h>
#include <assert.h>

BasicWindow::BasicWindow(const char* name1,const char* name2):BasicWidget(name2)
{
  _window = NULL;                // This class doesn't create the base window.

  assert(name1 != NULL);
  _windowName = XtNewString(name1);
}

BasicWindow::~BasicWindow(void)
{
  // Destroy the window, if it still exists.

  if(_window)
  {
    XtDestroyWidget(_window);
  }

  XtFree(_windowName);
}

void BasicWindow::rename(char* newName)
{
  XtFree(_windowName);
  _windowName = XtNewString(newName);

  assert(_window);
  XtVaSetValues(_window,
                XmNtitle,_windowName,
                NULL);

}

// Resize the window with new sizes.

void BasicWindow::resize(int width,int height)
{
  XtVaSetValues(_window,
                XmNwidth,     width,
                XmNmaxWidth,  width,
                XmNminWidth,  width,
                XmNheight,    height,
                XmNmaxHeight, height,
                XmNminHeight, height,
                NULL);
```

```
  XtVaSetValues(_widget,
                XmNwidth,  width,
                XmNheight, height,
                NULL);
}

const Widget BasicWindow::baseWindow(void)
{
  return _window;
}

// Realize, and popup the window  make sure it pops up on top of other windows.

void BasicWindow::manage(void)
{
  assert(_window);

  if(!XtIsRealized(_window))
  {
    XtRealizeWidget(_window);
  }

  XtPopup(_window,XtGrabNone);
  XMapRaised(theApplication->display(),XtWindow(_window));
}

void BasicWindow::unmanage(void)
{
  assert(_window);
  XtPopdown(_window);
}

void BasicWindow::iconify(void)
{
  assert(_window);

  // Set the widget to have an initial iconic state
  // in case the base widget has not yet been realized

  XtVaSetValues(_window,
                XmNiconic,TRUE,
                NULL);

  // If the widget has already been realized, iconify it

  if(XtIsRealized(_window))
  {
    XIconifyWindow(theApplication->display(),
                   XtWindow(_window),
                   0);
  }
}

// Create the base window and base widget for the window tree.

void BasicWindow::initialize(void)
{
  createShell();

  XtVaSetValues(_window,
                XmNtitle,          _windowName,
                XmNdeleteResponse, XmUNMAP,     // Change the default from
                NULL);                          // destroy to unmap.

  createWidget();

  if(!XtIsManaged(_widget))
  {
    XtManageChild(_widget);
  }
}
////////////////////////////////////////////////////////////////////////////
// UnsortedData.h : Support an array of integers and functions and operators to
//                  access the array.
////////////////////////////////////////////////////////////////////////////

#ifndef UNSORTEDDATA_H
```

```
#define UNSORTEDDATA_H

class UnsortedData
{
  private:

    int* _data;                 // Array of integers.
    int  _size;                 // Size of the array.
    int  _minValue;             // Minimum value in the array.
    int  _maxValue;             // Maximum value in the array.

  public:

    UnsortedData();
    ~UnsortedData(void);

    int           operator[](int);
    UnsortedData& operator=(UnsortedData);

    int size(void);
    int minValue(void);
    int maxValue(void);

    void setData(int,int);
    void copy(int*,int);
};

#endif
////////////////////////////////////////////////////////////////////////////
// UnsortedData.C : Support an array of integers and functions and operators to
//                  access the array.
////////////////////////////////////////////////////////////////////////////

#include "UnsortedData.h"
#include <assert.h>

// Constructor intializes size of the array to zero and creates an array of
// size 50.

UnsortedData::UnsortedData(void)
{
  _size = 0;
  _data = new int[50];
}

// Free the dynamically create array.

UnsortedData::~UnsortedData(void)
{
  delete[] _data;
}

void UnsortedData::copy(int data[],int size)
{
  assert(size <= 50);

  _size = size;

  _minValue =  data[0];
  _maxValue =  data[0];

  for(int i = 0;i < _size;i++)
  {
    _data[i] = data[i];

    if(_data[i] < _minValue)
    {
      _minValue = _data[i];
    }

    if(_data[i] > _maxValue)
    {
      _maxValue = _data[i];
    }
  }
}
```

```
int UnsortedData::size(void)
{
  return _size;
}

int UnsortedData::operator[](int index)
{
  assert(index < 50);
  return _data[index];
}

UnsortedData& UnsortedData::operator=(UnsortedData data)
{
  _size = data.size();

  for(int i = 0;i < _size;i++)
  {
    _data[i] = data[i];
  }

  _minValue = data.minValue();
  _maxValue = data.maxValue();

  return (*this);
}

int UnsortedData::minValue(void)
{
  return _minValue;
}

int UnsortedData::maxValue(void)
{
  return _maxValue;
}

void UnsortedData::setData(int index,int value)
{
  _data[index] = value;
}
////////////////////////////////////////////////////////////////////////////
// SortInfo.h : Data structure for sorting information.
////////////////////////////////////////////////////////////////////////////

#ifndef SORTINFO_H
#define SORTINFO_H

#ifndef TRUE

const int TRUE  = 1;
const int FALSE = 0;

#endif

class SortInfo
{
  private:

    int _index1,_index2;
    int _value1,_value2;
    int _isSwap;

  public:

    int index1(void);
    int index2(void);
    int value1(void);
    int value2(void);
    int isSwap(void);

    void setInfo(int,int,int,int,int);
};

#endif
////////////////////////////////////////////////////////////////////////////
// SortInfo.C : Data structure for sorting information.
////////////////////////////////////////////////////////////////////////////
```

```
#include "SortInfo.h"

int SortInfo::index1(void)
{
  return _index1;
}

int SortInfo::index2(void)
{
  return _index2;
}

int SortInfo::value1(void)
{
  return _value1;
}

int SortInfo::value2(void)
{
  return _value2;
}

void SortInfo::setInfo(int index1,int value1,int index2,int value2,int swap)
{
  _index1 = index1;
  _value1 = value1;

  _index2 = index2;
  _value2 = value2;

  _isSwap = swap;
}

int SortInfo::isSwap(void)
{
  return _isSwap;
}
////////////////////////////////////////////////////////////////////////////////
// Sort.h : Abstract class to implement sorting.
////////////////////////////////////////////////////////////////////////////////

#ifndef SORT_H
#define SORT_H

// Forward Reference.

class UnsortedData;
class SortInfo;

const int InsertionSortClass = 1;
const int ShellSortClass     = 2;
const int BubbleSortClass    = 3;
const int CombSortClass      = 4;
const int QuickSortClass     = 5;
const int SelectionSortClass = 6;

const int NoOfSorts          = 6;

extern char* const sortName[NoOfSorts];
extern int   const sortClass[NoOfSorts];

class Sort
{
  protected:

    // Data members to store unsorted data and sorting information.

    int _steps,
    int _size,_swaps,_comps;

    int*      _array;
    SortInfo* _info;

    void swap(int&,int&);

  public:
```

```
    Sort(void);
    virtual ~Sort(void);

    void reinit(void);

    // Virtual member function to be implemented by derived classes.

    virtual SortInfo* doTheSorting(UnsortedData) = 0;

    // Functions to return data and sorting information.

    int steps(void);
    int size(void);
    int swaps(void);
    int comps(void);
};

#endif
////////////////////////////////////////////////////////////////////////////
// Sort.C : Abstract class to implement sorting.
////////////////////////////////////////////////////////////////////////////

#include "Sort.h"
#include "UnsortedData.h"
#include "SortInfo.h"

#include <stdlib.h>

char* const sortName[NoOfSorts]   = {"Insertion Sort",
                                     "Shell Sort",
                                     "Bubble Sort",
                                     "Comb Sort",
                                     "Quick Sort",
                                     "Selection Sort"};

int   const sortClass[NoOfSorts] = {1,2,3,4,5,6};

// Constructor initializes all data members to NULL.

Sort::Sort(void)
{
  _steps = _size = _comps = _swaps = 0;
  _info  = NULL;
  _array = NULL;
}

Sort::~Sort(void)
{
}

// Reinitialize data members to NULL.

void Sort::reinit(void)
{
  delete []_info;
  delete []_array;

  _steps = _size = _swaps = _comps = 0;
}

void Sort::swap(int& a1,int& a2)
{
  int temp;

  temp = a1;
  a1   = a2;
  a2   = temp;
}

int Sort::steps(void)
{
  return _steps;
}

int Sort::size(void)
{
  return _size;
```

```
}

int Sort::comps(void)
{
  return _comps;
}

int Sort::swaps(void)
{
  return _swaps;
}
////////////////////////////////////////////////////////////////////////////
// CmdInterface.h : Abstract class to support a command interface object.
////////////////////////////////////////////////////////////////////////////

#ifndef CMDINTERFACE_H
#define CMDINTERFACE_H

#include "BasicWidget.h"

class CmdInterface:public BasicWidget
{
  private:

    char* _cmdName;               // Command name.

  protected:

    XmFontList _fontList;         // Default font to be used by derived objects.

  public:

    CmdInterface(const char*);
    virtual ~CmdInterface(void);

    // Activate and deactivate commands.

    virtual void activate(void);
    virtual void deactivate(void);

    // Static member callback function for the command.

    static void CmdInterfaceCallback(Widget,XtPointer,XtPointer);

    // Virtual member function to be implemented by the derived classes.

    virtual void execute(Widget,XtPointer) = 0;
};

#endif
////////////////////////////////////////////////////////////////////////////
// CmdInterface.C : Abstract class support a command interface object.
////////////////////////////////////////////////////////////////////////////

#include "CmdInterface.h"
#include "Application.h"

// Constructor creates a default font to be used by the derived objects.

CmdInterface::CmdInterface(const char* name):BasicWidget(name)
{
  char* fontName    = "-adobe-helvetica-bold-r-*-14-*";
  XFontStruct* font = XLoadQueryFont(theApplication->display(),fontName);

  _fontList         = XmFontListCreate(font,XmSTRING_DEFAULT_CHARSET);

  _cmdName          = XtNewString(name);
}

// Destructor frees the command name and the font list.

CmdInterface::~CmdInterface(void)
{
  XtFree(_cmdName);
  XmFontListFree(_fontList);
}
```

```
void CmdInterface::activate(void)
{
  if(_widget)
  {
    XtSetSensitive(_widget,TRUE);
  }
}

void CmdInterface::deactivate(void)
{
  if(_widget)
  {
    XtSetSensitive(_widget,FALSE);
  }
}

void CmdInterface::CmdInterfaceCallback(Widget w,XtPointer clientData,
                                                    XtPointer callData)
{
  // Cast the clientData to the expected object type

  CmdInterface* obj = (CmdInterface*)clientData;

  // Call the corresponding member function

  obj->execute(w,callData);
}
////////////////////////////////////////////////////////////////////////
// CmdButton.h : Abstract class support a pushbutton as the user-interface
//               object. This is still an abstract class because the execute()
//               function is not implemented yet.
////////////////////////////////////////////////////////////////////////

#ifndef CMDBUTTON_H
#define CMDBUTTON_H

#include "CmdInterface.h"

class CmdButton:public CmdInterface
{
  public:

    CmdButton(const char*,Widget,int);
    virtual ~CmdButton(void){}
};

#endif
////////////////////////////////////////////////////////////////////////
// CmdButton.C : Abstract class to support a pushbutton as the user-interface
//               object.
////////////////////////////////////////////////////////////////////////

#include "CmdButton.h"
#include <Xm/PushBG.h>

// Constructor creates a pushbutton and registers the callback function.

CmdButton::CmdButton(const char* name,Widget parent,int active):
                                                    CmdInterface(name)
{
  _widget = XtVaCreateWidget(name,
                             xmPushButtonGadgetClass,parent,
                             XmNsensitive,active,
                             XmNfontList, _fontList,
                             NULL);

  XtAddCallback(_widget,
                XmNactivateCallback,&CmdButton::CmdInterfaceCallback,
                (XtPointer)this);
}
////////////////////////////////////////////////////////////////////////
// CmdToggle.h : Abstract class to support a togglebutton as the user-interface
//               object. This is still an abstract class because the execute()
//               is not implemented yet.
////////////////////////////////////////////////////////////////////////

#ifndef CMDTOGGLE_H
```

```
#define CMDTOGGLE_H

#include "CmdInterface.h"

class CmdToggle:public CmdInterface
{
  public:

    CmdToggle(const char*,Widget,int);
    virtual ~CmdToggle(void){}

    // Check to see if the togglebutton is selected.

    int  isSelected(void);

    // Uncheck togglebutton.

    virtual void deSelect(void);
};

#endif
////////////////////////////////////////////////////////////////////////////////
// CmdToggle.C : Abstract class to support a togglebutton as the user-interface
//               object.
////////////////////////////////////////////////////////////////////////////////

#include "CmdToggle.h"

#include <Xm/ToggleBG.h>

// Constructor creates the togglebutton and registers callback functions.

CmdToggle::CmdToggle(const char* name,Widget parent,int active):
                                                        CmdInterface(name)
{
  _widget = XtVaCreateWidget(name,
                             xmToggleButtonGadgetClass,
                             parent,
                             XmNsensitive,      active,
                             XmNindicatorType,  XmN_OF_MANY,
                             XmNvisibleWhenOff, True,
                             XmNfontList,       _fontList,
                             NULL);

  XtAddCallback(_widget,
                XmNvalueChangedCallback,&CmdToggle::CmdInterfaceCallback,
                (XtPointer)this);
}

// See if the togglebutton is checked.

int CmdToggle::isSelected(void)
{
  return XmToggleButtonGadgetGetState(_widget);
}

// Unselect the togglebutton.

void CmdToggle::deSelect(void)
{
  XmToggleButtonGadgetSetState(_widget,False,False);
}
////////////////////////////////////////////////////////////////////////////////
// BubbleSort.h : Implement BubbleSort.
////////////////////////////////////////////////////////////////////////////////

#ifndef BUBBLESORT_H
#define BUBBLESORT_H

#include "Sort.h"

class BubbleSort:public Sort
{
  public:

    BubbleSort(void):Sort(){}
    virtual ~BubbleSort(void){}
```

```
    virtual SortInfo* doTheSorting(UnsortedData);
};

#endif
////////////////////////////////////////////////////////////////////////////////
// BubbleSort.C : Implement Bubblesort.
////////////////////////////////////////////////////////////////////////////////

#include "BubbleSort.h"
#include "SortInfo.h"
#include "UnsortedData.h"

SortInfo* BubbleSort::doTheSorting(UnsortedData data)
{
  int       i,j;
  int       flag;

  _size  = data.size();
  _array = new int[_size];
  _info  = new SortInfo[_size * _size];

  for(i = 0;i < _size;i++)
  {
    _array[i] = data[i];
  }

  for(i = 0,flag = TRUE;(i < _size) && (flag);i++)
  {
    flag = FALSE;

    // This step is a comparison operation.

    for(j = 0;j < (_size - i);j++)
    {
      _info[_steps].setInfo(j,
                            _array[j],
                            j + 1,
                            _array[j + 1],
                            FALSE);
      _steps++;
      _comps++;

      if(_array[j] > _array[j+1])
      {
        flag = TRUE;

        // Swap them.

        swap(_array[j],_array[j+1]);

        // This step is a swap operation.

        _info[_steps].setInfo(j,
                              _array[j],
                              j + 1,
                              _array[j + 1],
                              TRUE);
        _steps++;
        _swaps++;
      }
    }
  }

  return _info;
}
////////////////////////////////////////////////////////////////////////////////
// CombSort.h : Implement Combsort.
////////////////////////////////////////////////////////////////////////////////

#ifndef COMBSORT_H
#define COMBSORT_H

#include "Sort.h"

class CombSort:public Sort
{
  public:
```

```
    CombSort(void):Sort(){}
    virtual ~CombSort(void){}

    virtual SortInfo* doTheSorting(UnsortedData);
};

#endif
////////////////////////////////////////////////////////////////////////////
// CombSort.C : Implement Combsort.
////////////////////////////////////////////////////////////////////////////

#include "CombSort.h"
#include "SortInfo.h"
#include "UnsortedData.h"

const float SHRINKFACTOR = 1.3;

SortInfo* CombSort::doTheSorting(UnsortedData data)
{
  int       gap,i,j;
  int       switches,top;

  _size  = gap = data.size();
  _array = new int[_size];
  _info  = new SortInfo[_size * _size];

  for(i = 0;i < _size;i++)
  {
    _array[i] = data[i];
  }

  do
  {
    gap = (int)((float)gap/SHRINKFACTOR);

    switch(gap)
    {
      case 0:                   // The smallest gap is 1 - BubbleSort.

        gap = 1;
        break;

      case 9:                   // This is what makes this a Combsort11.

      case 10:

        gap = 11;
        break;

      default:

        break;
    }

    switches = 0;               // Dirty pass flag.

    top = _size - gap;

    for(i = 0;i < top;i++)
    {
      j = i + gap;

      // This step is a comparison operation.

      _info[_steps].setInfo(i,
                            _array[i],
                            j,
                            _array[j],
                            FALSE);
      _steps++;
      _comps++;

      if(_array[i] > _array[j])
      {
        swap(_array[i],_array[j]);
        switches++;
```

```
        // This step is a swap operation.

        _info[_steps].setInfo(i,
                              _array[i],
                              j,
                              _array[j],
                              TRUE);
        _steps++;
        _swaps++;
      }
    }                                 // End of pass.

  }while(switches || (gap > 1));      // Like BubblesSort and ShellSort, check
                                      // for a clean pass.

  return _info;
}
////////////////////////////////////////////////////////////////////////////////
// InsertionSort.h : Implement Insertionsort.
////////////////////////////////////////////////////////////////////////////////

#ifndef INSERTIONSORT_H
#define INSERTIONSORT_H

#include "Sort.h"

class InsertionSort:public Sort
{
  public:

    InsertionSort(void):Sort(){}
    virtual ~InsertionSort(void){}

    virtual SortInfo* doTheSorting(UnsortedData);
};

#endif
////////////////////////////////////////////////////////////////////////////////
// InsertionSort.C : Implement Insertionsort.
////////////////////////////////////////////////////////////////////////////////

#include "InsertionSort.h"
#include "SortInfo.h"
#include "UnsortedData.h"

SortInfo* InsertionSort::doTheSorting(UnsortedData data)
{
  int       i,j,temp;

  _size  = data.size();
  _array = new int[_size];
  _info  = new SortInfo[_size * _size];

  for(i = 0;i < _size;i++)
  {
    _array[i] = data[i];
  }

  for(i = (_size - 2);i >= 0;i--)
  {
    j    = i + 1;
    temp = _array[i];

    // Insert _array[i] in its sorted position among
    // _array[i+1], _array[i+2], ..., _array[_size - 1].

    while(temp > _array[j])
    {
      // This step is a comparison operation.

      _info[_steps].setInfo(i,
                            _array[i],
                            j,
                            _array[j],
                            FALSE);
      _steps++;
      _comps++;
```

```
      _array[j - 1] = _array[j];

      // This step is a swap operation.

      _info[_steps].setInfo(j - 1,
                            _array[j - 1],
                            j,
                            _array[j],
                            TRUE);
      _steps++;
      _swaps++;

      j++;
    }

    _array[j - 1] = temp;

    // This is a swap operation again.

    _info[_steps].setInfo(j - 1,
                          _array[j - 1],
                          i,
                          _array[i],
                          TRUE);
    _steps++;
    _swaps++;
  }

  return _info;
}
////////////////////////////////////////////////////////////////////////////////
// QuickSort.h : Implement Quicksort.
////////////////////////////////////////////////////////////////////////////////

#ifndef QUICKSORT_H
#define QUICKSORT_H

#include "Sort.h"

class QuickSort:public Sort
{
  private:

    void qSort(int,int);

  public:

    QuickSort(void):Sort(){}
    virtual ~QuickSort(void){}

    virtual SortInfo* doTheSorting(UnsortedData);
};

#endif
////////////////////////////////////////////////////////////////////////////////
// QuickSort.C : Implement Quicksort.
////////////////////////////////////////////////////////////////////////////////

#include "QuickSort.h"
#include "SortInfo.h"
#include "UnsortedData.h"

SortInfo* QuickSort::doTheSorting(UnsortedData data)
{
   _size  = data.size();
   _array = new int[_size];
   _info  = new SortInfo[_size * _size];

  for(int i = 0;i < _size;i++)
  {
    _array[i] = data[i];
  }

  if(_size > 0)
  {
    qSort(0,(_size - 1));
  }
```

```
  return _info;
}

void QuickSort::qSort(int First,int Last)
{
  int i,j,Pivot;

  if(First < Last)
  {
    Pivot = _array[First];
    i     = First;
    j     = Last;

    while(i < j)
    {
      while((_array[i] <= Pivot) && (i < Last))
      {
        _info[_steps].setInfo(i,
                              _array[i],
                              First,
                              _array[First],
                              FALSE);
        _steps++;
        _comps++;

        i++;
      }

      while((_array[j] >= Pivot) && (j > First))
      {
        // This step is a comparison operation.

        _info[_steps].setInfo(First,
                              _array[First],
                              j,
                              _array[j],
                              FALSE),
        _steps++;
        _comps++;

        j--;
      }

      if(i < j)
      {
        swap(_array[j],_array[i]);

        // This step is a swap operation.

        _info[_steps].setInfo(i,
                              _array[i],
                              j,
                              _array[j],
                              TRUE);
        _steps++;
        _swaps++;
      }
    }

    // Interchange _array[j] and _array[First].

    swap(_array[j],_array[First]);

    // This is a swap operation.

    _info[_steps].setInfo(First,
                          _array[First],
                          j,
                          _array[j],
                          TRUE);
    _steps++;
    _swaps++;

    // Call qSort recursively.

    qSort(First,(j - 1));
    qSort((j + 1),Last);
```

```
  }
}
////////////////////////////////////////////////////////////////////////////////
// SelectionSort.h : Implement Selectionsort.
////////////////////////////////////////////////////////////////////////////////

#ifndef SELECTIONSORT_H
#define SELECTIONSORT_H

#include "Sort.h"

class SelectionSort:public Sort
{
  public:

    SelectionSort(void):Sort(){}
    virtual ~SelectionSort(void){}

    virtual SortInfo* doTheSorting(UnsortedData);
};

#endif
////////////////////////////////////////////////////////////////////////////////
// SelectionSort.C : Implement Selectionsort.
////////////////////////////////////////////////////////////////////////////////

#include "SelectionSort.h"
#include "SortInfo.h"
#include "UnsortedData.h"

SortInfo* SelectionSort::doTheSorting(UnsortedData data)
{
  int       i,j;
  int       min;

  _size  = data.size();
  _array = new int[_size];
  _info  = new SortInfo[_size * _size];

  for(i = 0;i < _size;i++)
  {
    _array[i] = data[i];
  }

  for(i = 0;i < _size;i++)
  {
    min = i;
    for(j = i+1;j < _size;j++)
    {
      // This step is a comparison operation.

      _info[_steps].setInfo(j,
                            _array[j],
                            min,
                            _array[min],
                            FALSE);
      _steps++;
      _comps++;

      if(_array[j] < _array[min])
      {
        min = j;
      }
    }
    // Swap _array[min] and _array[i].

    swap(_array[min],_array[i]);

    // This step is a swap operation.

    _info[_steps].setInfo(min,
                          _array[min],
                          i,
                          _array[i],
                          TRUE);
    _steps++;
    _swaps++;
```

```
  }

  return _info;
}
////////////////////////////////////////////////////////////////////////////
// ShellSort.h : Implement Shellsort.
////////////////////////////////////////////////////////////////////////////

#ifndef SHELLSORT_H
#define SHELLSORT_H

#include "Sort.h"

class ShellSort:public Sort
{
  public:

    ShellSort(void):Sort(){}
    virtual ~ShellSort(void){}

    virtual SortInfo* doTheSorting(UnsortedData);
};

#endif
////////////////////////////////////////////////////////////////////////////
// ShellSort.C : Implement Shellsort.
////////////////////////////////////////////////////////////////////////////

#include "ShellSort.h"
#include "SortInfo.h"
#include "UnsortedData.h"

SortInfo* ShellSort::doTheSorting(UnsortedData data)
{
  int       i,temp,search;
  int       shellSize;
  int       kSort,kSort2,kSort1;

  _size  = shellSize = data.size();
  _array = new int[_size];
  _info  = new SortInfo[_size * _size];

  for(i = 0;i < _size;i++)
  {
    _array[i] = data[i];
  }

  while(shellSize > 1)
  {
    // Find value of 'k' to break sort process into sub_arrays by the
    // following rule:
    // kSort = kSort1 * 4 - kSort2 * 3

    kSort2 = 0;
    kSort1 = 1;

    while((kSort = kSort1 * 4 - kSort2 * 3) < shellSize)
    {
      kSort2 = kSort1;
      kSort1 = kSort;
    }

    kSort = kSort1;

    // Perform InsertionSort on 'k' sort sub_arrays.

    for(i = 0;i < _size;i++)
    {
      temp = _array[i];

      search = i - kSort;

      while((temp < _array[search]) && (search >= 0))
      {
        // This step is a comparison operation.

        _info[_steps].setInfo(i,
```

```
                                _array[i],
                                search,
                                _array[search],
                                FALSE);
            _steps++;
            _comps++;

            _array[search + kSort] = _array[search];

            // This step is a swap operation.

            _info[_steps].setInfo((search + kSort),
                                _array[search + kSort],
                                search,
                                _array[search],
                                TRUE);
            _steps++;
            _swaps++;

            search = search - kSort;
          }

          if(i != (search + kSort))
          {
            _array[search + kSort] = temp;

            // This step is a swap operation.

            _info[_steps].setInfo((search + kSort),
                                _array[search + kSort],
                                i,
                                _array[i],
                                TRUE);
            _steps++;
            _swaps++;
          }
        }

        shellSize = kSort;
      }

      return _info;
}
////////////////////////////////////////////////////////////////////////////////
// Canvas.h : Support a pixmap and other drawing functions.
////////////////////////////////////////////////////////////////////////////////

#ifndef CANVAS_H
#define CANVAS_H

#include <Xm/Xm.h>

const int Font17 = 17;
const int Font14 = 14;
const int Font8  = 8;

class Canvas
{

  private:

    Pixmap _pixmap;

    GC _gcDefault;
    GC _gcClean;
    GC _gcFont17,_gcFont14,_gcFont8;

    int _width,_height;

    Widget _parent;

  public:

    Canvas(Widget);
    ~Canvas(void);

    // Create a new pixmap.
```

```
    void create(void);

    // Clear the pixmap.

    void clean();

    // Copy the pixmap to the drawing area.

    void copy(void);

    // Drawing functions.

    void drawRectangle(int,int,int,int);
    void drawLine(int,int,int,int);
    void drawString(int,int,char*,int);
    void drawCircle(int,int,int);
    void fillRectangle(int,int,int,int);
};

#endif
////////////////////////////////////////////////////////////////////////////
// Canvas.C : Support a pixmap to draw on. Support functions to draw on the
//            pixmap, to clean the pixmap and copy the pixmap to the parent.
////////////////////////////////////////////////////////////////////////////

#include "Canvas.h"
#include "Application.h"
#include "PopupDialog.h"

#include <string.h>
#include <stdio.h>
#include <assert.h>

// Constructor creates various graphics contexts that may be needed to draw.

Canvas::Canvas(Widget parent)
{
  assert(parent);
  _parent = parent;

  _pixmap = NULL;

  XGCValues gcv;

  XtVaGetValues(_parent,
                XmNbackground, &gcv.foreground,
                NULL);

  _gcClean = XtGetGC(_parent,
                     GCForeground,&gcv);

  XFontStruct* font = NULL;

  font     = XLoadQueryFont(theApplication->display(),"*-i-*-17-*");
  gcv.font = font->fid;

  _gcFont17 = XtGetGC(_parent,
                      GCFont,&gcv);

  font      = XLoadQueryFont(theApplication->display(),"*-i-*-14-*");
  gcv.font  = font->fid;

  _gcFont14 = XtGetGC(_parent,
                      GCFont,&gcv);

  font      = XLoadQueryFont(theApplication->display(),"*-times-*-8-*");
  gcv.font  = font->fid;

  _gcFont8  = XtGetGC(_parent,
                      GCFont,&gcv);

  XFreeFont(theApplication->display(),font);

  _gcDefault = XtGetGC(_parent,
                       0,NULL);
}
```

```
// Release all the graphics contexts created. Free the pixmap if it still
// exists.

Canvas::~Canvas(void)
{
  XtReleaseGC(_parent,_gcClean);

  XtReleaseGC(_parent,_gcFont17);
  XtReleaseGC(_parent,_gcFont14);
  XtReleaseGC(_parent,_gcFont8);

  XtReleaseGC(_parent,_gcDefault);

  if(_pixmap)
  {
    XFreePixmap(theApplication->display(),_pixmap);
  }
}

// Pixmaps can't be resized. So destroy the old one and create a new one of
// the size needed.

void Canvas::create(void)
{
  if(_pixmap)
  {
    XFreePixmap(theApplication->display(),_pixmap);
  }

  XtVaGetValues(_parent,
                XmNwidth, &_width,
                XmNheight,&_height,
                NULL);

  _pixmap = XCreatePixmap(theApplication->display(),
                          DefaultRootWindow(theApplication->display()),
                          _width,
                          _height,
                          DefaultDepthOfScreen(XtScreen(_parent)));
}

// Clean the pixmap with the background color of the parent.

void Canvas::clean(void)
{
  XFillRectangle(theApplication->display(),
                 _pixmap,
                 _gcClean,
                 0,
                 0,
                 _width,
                 _height);
}

// Pixmap don't appear on the screen directly. So copy the pixmap to the
// parent drawing area.

void Canvas::copy(void)
{
  if(DefaultDepthOfScreen(XtScreen(_parent)) == 1)
  {
    XCopyArea(XtDisplay(_parent),
              _pixmap,
              XtWindow(_parent),
              _gcDefault,
              0,
              0,
              _width,
              _height,
              0,
              0);
  }

  else
  {
    XCopyPlane(XtDisplay(_parent),
               _pixmap,
```

```
                XtWindow(_parent),
                _gcDefault,
                0,
                0,
                _width,
                _height,
                0,
                0,
                1);
  }
}

// Drawing functions.

void Canvas::drawRectangle(int x,int y,int width,int height)
{
  XDrawRectangle(XtDisplay(_parent),
                 _pixmap,
                 _gcDefault,
                 x,
                 y,
                 width,
                 height);
}

void Canvas::fillRectangle(int x,int y,int width,int height)
{
  XFillRectangle(XtDisplay(_parent),
                 _pixmap,
                 _gcDefault,
                 x,
                 y,
                 width,
                 height);
}

void Canvas::drawLine(int x1,int y1,int x2,int y2)
{
  XDrawLine(theApplication->display(),
            _pixmap,
            _gcDefault,
            x1,
            y1,
            x2,
            y2);
}

void Canvas::drawString(int x1,int y1,char* text,int fontName)
{
  switch(fontName)
  {
    case Font17:

      XDrawString(theApplication->display(),
                  _pixmap,
                  _gcFont17,
                  x1,
                  y1,
                  text,
                  strlen(text));
      break;

    case Font14:

      XDrawString(theApplication->display(),
                  _pixmap,
                  _gcFont14,
                  x1,
                  y1,
                  text,
                  strlen(text));
      break;

    case Font8:

     XDrawString(theApplication->display(),
                 _pixmap,
```

```
                    _gcFont8,
                    x1,
                    y1,
                    text,
                    strlen(text));

      break;

    default:

      XDrawString(theApplication->display(),
                    _pixmap,
                    _gcDefault,
                    x1,
                    y1,
                    text,
                    strlen(text));
    break;
  }
}

void Canvas::drawCircle(int x,int y,int diameter)
{
  XDrawArc(theApplication->display(),
           _pixmap,
           _gcDefault,
           x,
           y,
           diameter,
           diameter,
           0,
           64 * 360);
}
////////////////////////////////////////////////////////////////////////////
// DrawWindow.h : Abstract class to support drawing.
//                This class provides a window and a drawing area to copy
//                pixmaps onto.
////////////////////////////////////////////////////////////////////////////

#ifndef DRAWWINDOW_H
#define DRAWWINDOW_H

#include "BasicWindow.h"

class DrawWindow:public BasicWindow
{
  private:

    // Static member callback functions.

    static void exposeCallback(Widget,XtPointer,XtPointer);
    static void okCallback(Widget,XtPointer,XtPointer);

  protected:

    Widget _okButton;                // Pushbutton to close the window.
    Widget _drawArea;                // Drawing area widget.

    // Callback functions implemented

    virtual void resize(int,int);
    virtual void ok(void);

    // Callback functions implemented by the derived classes.

    virtual void expose(void) = 0;

    virtual void initialize(void);
    virtual void manage(void);

    int _color;

  public:

    DrawWindow(const char*);
    virtual ~DrawWindow(void){}
```

```
    // Create a shell and a widget for the window to exist.

    virtual void createShell(void);
    virtual void createWidget(void);
};

#endif
////////////////////////////////////////////////////////////////////////////
// DrawWindow.h : Abstract class to support drawing.
////////////////////////////////////////////////////////////////////////////

#include "DrawWindow.h"
#include "Application.h"

#include <Xm/Form.h>
#include <Xm/PushB.h>
#include <Xm/DrawingA.h>
#include <assert.h>

DrawWindow::DrawWindow(const char* name):BasicWindow(name,"DrawingArea")
{
}

// Create a window.

void DrawWindow::createShell(void)
{
  _window = XtVaCreatePopupShell(_windowName,
                                 topLevelShellWidgetClass,
                                 theApplication->toplevel(),
                                 XmNtitle,        _windowName,
                                 XmNx,            200,
                                 XmNy,            100,
                                 NULL);
}

// Create a form widget.

void DrawWindow::createWidget(void)
{
  _widget = XtVaCreateWidget("DrawWindow",
                             xmFormWidgetClass,_window,
                             NULL);
}

// Create a drawing area widget and position the pushbutton and drawing area
// on the form.

void DrawWindow::initialize(void)
{
  BasicWindow::initialize();

  _drawArea = XtVaCreateWidget("Draw",
                               xmDrawingAreaWidgetClass,
                               _widget,
                               NULL);

  XtAddCallback(_drawArea,
                XmNexposeCallback,&DrawWindow::exposeCallback,
                (XtPointer)this);

  _okButton = XtVaCreateWidget("Close",
                               xmPushButtonWidgetClass,
                               _widget,
                               NULL);
  XtAddCallback(_okButton,
                XmNactivateCallback,&DrawWindow::okCallback,
                (XtPointer)this);
}

// Expose callback function.

void DrawWindow::exposeCallback(Widget,XtPointer clientData,XtPointer)
{
  DrawWindow* obj = (DrawWindow*)clientData;
  obj->expose();
}
```

```
// Ok callback function.

void DrawWindow::okCallback(Widget,XtPointer clientData,XtPointer)
{
  DrawWindow* obj = (DrawWindow*)clientData;
  obj->ok();
}

void DrawWindow::ok(void)
{
  BasicWindow::unmanage();
}

// Resize the window to the width and height and position the drawing area
// widget and the Ok button.

void DrawWindow::resize(int width,int height)
{
  BasicWindow::resize(width,(height + 25));

  XtVaSetValues(_drawArea,
                XmNwidth,            width,
                XmNheight,           height,
                XmNtopAttachment,    XmATTACH_FORM,
                XmNleftAttachment,   XmATTACH_FORM,
                XmNrightAttachment,  XmATTACH_FORM,
                XmNbottomAttachment, XmATTACH_NONE,
                NULL);

  XtVaSetValues(_okButton,
                XmNtopAttachment,    XmATTACH_NONE,
                XmNleftAttachment,   XmATTACH_FORM,
                XmNrightAttachment,  XmATTACH_FORM,
                XmNbottomAttachment, XmATTACH_FORM,
                NULL);
}

void DrawWindow::manage(void)
{
  BasicWindow::manage();

  XtManageChild(_drawArea);
  XtManageChild(_okButton);
}
////////////////////////////////////////////////////////////////////////////
// DisplayWindow.h : Support display window.
//                   This class handles the actual display. Based on the sorting
//                   information, rectangles are drawn on a pixmap which is
//                   later copied to the drawing area. For a smooth animation,
//                   a technique called double buffering is used.
//
//                   The class has two pixmaps (Class Canvas implements pixmpas)
//                   of which only one is visible at any time. All the objects
//                   that appear on the next frame of the display are drawn on
//                   the hidden canvas. When the entire scene has been rendered,
//                   the hidden pixmap is displayed. The previously hidden
//                   pixmap is hidden, erased, and made available for drawing
//                   the next frame.
////////////////////////////////////////////////////////////////////////////

#ifndef DISPLAYWINDOW_H
#define DISPLAYWIDNOW_H

#include "DrawWindow.h"
#include "UnsortedData.h"

// Forward reference.

class SortList;
class SortInfo;
class Canvas;

class DisplayWindow:public DrawWindow
{
  private:

    SortList* _sort;            // Sorting methods implemented.
```

```
    // Data members to handle sorting information.

    int _swaps, _comps;
    int _width, _height;
    int _baseWidth, _baseHeight,
    int _gap;

    UnsortedData _data;

    // Two canvases needed for the animation.

    Canvas* _front;
    Canvas* _back;

    void    swap(Canvas*&,Canvas*&);

    // Expose Callback function..

    virtual void expose(void);

  public:

    DisplayWindow(const char*,SortList* sort = NULL);
    ~DisplayWindow(void);

    void reinit(UnsortedData);

    // Functions to draw.

    void draw(SortInfo);
    void draw(void);

    // Copy the next frame to the drawing area.

    void nextFrame(void);

    virtual void ok(void);
};

#endif
////////////////////////////////////////////////////////////////////////////////
// DisplayWindow.C : Support display window.
//                   This class handles the actual display.
////////////////////////////////////////////////////////////////////////////////

#include "DisplayWindow.h"
#include "Application.h"
#include "Canvas.h"
#include "SortInfo.h"
#include "SortList.h"

#include <stdio.h>
#include <assert.h>

DisplayWindow::DisplayWindow(const char* name,SortList* sort):DrawWindow(name)
{
  _sort = sort;

  _baseWidth  = 12;
  _baseHeight = 5;

  // Create the display window.

  DrawWindow::initialize();

  _front = new Canvas(_drawArea);
  _back  = new Canvas(_drawArea);
}

DisplayWindow::~DisplayWindow(void)
{
  delete _front;
  delete _back;
}

void DisplayWindow::reinit(UnsortedData data)
{
```

```
  _swaps = _comps = 0;
  _data  = data;

  _height = 20 + _baseHeight * _data.maxValue();
  _width  = 10 + _baseWidth  * _data.size();

  DrawWindow::resize(_width,_height);
  DrawWindow::manage();

  assert(XtIsRealized(_window));

  // Pixmaps can't be resized, so create a new one.

  _front->create();
  _back->create();

  _front->clean();
  _back->clean();
}

// Render the next scene based on sorting information.

void DisplayWindow::draw(SortInfo info)
{
  _data.setData(info.index1(),info.value1());
  _data.setData(info.index2(),info.value2());

  assert(XtIsRealized(_window));

  // If swap, partially fill the rectangles that are being swappped.

  if(info.isSwap())
  {
    _swaps++;

    _back->fillRectangle(5 + info.index1() * _baseWidth,
                         _height - 20 - info.value1() * _baseHeight,
                         _baseWidth,
                         info.value1() * _baseHeight / 2);

    _back->fillRectangle(5 + info.index2() * _baseWidth,
                         _height - 20 - info.value2() * _baseHeight,
                         _baseWidth,
                         info.value2() * _baseHeight / 2);
  }

  // If compare, completely fill the rectangles that are being compared.

  else
  {
    _comps++;

    _back->fillRectangle(5 + info.index1() * _baseWidth,
                         _height - 20 - info.value1() * _baseHeight,
                         _baseWidth,
                         info.value1() * _baseHeight);

    _back->fillRectangle(5 + info.index2() * _baseWidth,
                         _height - 20 - info.value2() * _baseHeight,
                         _baseWidth,
                         info.value2() * _baseHeight);
  }

  int  offset;
  char str[20];

  // Draw all the rectangles representing the data on the canvas.

  for(int i = 0;i < _data.size();i++)
  {
    _back->drawRectangle(5 + i * _baseWidth,
                         _height - 20 - _data[i] * _baseHeight,
                         _baseWidth,
                         _data[i] * _baseHeight);

    sprintf(str,"%d",_data[i]);
```

```
    if(_data[i] > 9)
    {
      offset = 7;
    }
    else
    {
      offset = 10;
    }

    // Draw the numbers the rectangles represent within the rectangles.

    _back->drawString(offset + i * _baseWidth,
                      _height - 22 - _data[i] * _baseHeight +
                                                  _data[i] * _baseHeight / 2,
                      str,
                      Font8);
  }

  // Draw the current comparisons and swaps on the canvas

  if((_width > 224) || (_height > 99))
  {
    sprintf(str,"Comps : %d",_comps);
    _back->drawString(10,20,str,Font14);

    sprintf(str,"Swaps : %d",_swaps);
    if(_width > 224)
    {
      _back->drawString(125,20,str,Font14);
    }
    else
    {
      _back->drawString(10,40,str,Font14);
    }
  }
}

// The initial unsorted data is drawn on the canvas.

void DisplayWindow::draw(void)
{
  int  offset;
  char str[20];

  assert(XtIsRealized(_window));

  // Draw the rectangles representing the data on the canvas.

  for(int i = 0;i < _data.size();i++)
  {
    _back->drawRectangle(5 + i * _baseWidth,
                         _height - 20 - _data[i] * _baseHeight,
                         _baseWidth,
                         _data[i] * _baseHeight);

    sprintf(str,"%d",_data[i]);

    if(_data[i] > 9)
    {
      offset = 7;
    }
    else
    {
      offset = 10;
    }

    // Draw the numbers the rectangles represent within the rectangles.

    _back->drawString(offset + i * _baseWidth,
                      _height - 22 - _data[i] * _baseHeight +
                                                  _data[i] * _baseHeight / 2,
                      str,
                      Font8);
  }

  if((_width > 224) || (_height > 99))
  {
```

```
    sprintf(str,"Comps : %d",_comps);
    _back->drawString(10,20,str,Font14);

    sprintf(str,"Swaps : %d",_swaps);
    if(_width > 224)
    {
      _back->drawString(125,20,str,Font14);
    }
    else
    {
      _back->drawString(10,40,str,Font14);
    }
  }

  XBell(theApplication->display(),-50);
}

void DisplayWindow::expose(void)
{
  assert(_front != NULL);
  _front->copy();
}

void DisplayWindow::nextFrame(void)
{
  swap(_front,_back);

  _front->copy();
  _back->clean();

  expose();
}

// Swap the canvases.

void DisplayWindow::swap(Canvas*& front,Canvas*& back)
{
  Canvas* tmp;

  tmp   = front;
  front = back;
  back  = tmp;
}

void DisplayWindow::ok(void)
{
  DrawWindow::ok();

  if(_sort)
  {
    _sort->deSelect();
  }
}
////////////////////////////////////////////////////////////////////////////////
// GraphWindow.h : Support a canvas to draw graph.
//                 Handle expose event.
//                 Functions to draw specific graphs.
////////////////////////////////////////////////////////////////////////////////

#ifndef GRAPHWINDOW_H
#define GRAPHWINDOW_H

#include "DrawWindow.h"

// Forward references.

class SortInfo;
class Canvas;

class GraphWindow:public DrawWindow
{
  private:

    Canvas* _graph;             // Canvas object.

    // Expose callback function.
```

```
                        300,
                        str,
                        Font14);
  }

  _graph->drawRectangle(70,30,150,100);
  _graph->drawString(120,50,"LEGEND",Font14);
  _graph->drawLine(70,60,220,60);
  _graph->drawRectangle(75,70,5,5);
  _graph->drawString(100,80,"Swaps",Font14);
  _graph->drawCircle(75,90,5);
  _graph->drawString(100,100."Comparisons",Font14);
  _graph->drawLine(75,110,80,110);
}

// "n^2 vs n" graph.

void GraphWindow::drawNSquareGraph(void)
{
  assert(XtIsRealized(_window));

  for(int i = 0;i < 50;i++)
  {
    _graph->drawLine(5 * i + 70,
                     280 - (int)(i * i * 0.10),
                     5 * (i + 1) + 70,
                     280 - (int)((i + 1) * (i + 1) * 0.10));
  }

  char str[4];
  for(i = 0;i <= 50;i+=10)
  {
    _graph->drawLine(65,
                     280 - (int)(i * i * 0.10),
                     70,
                     280 - (int)(i * i * 0.10));

    sprintf(str,"%d",i * i);

    _graph->drawString(5,
                       280 - (int)(i * i * 0.10),
                       str,
                       Font14);
  }

  _graph->drawString(100,
                     120,
                     "n^2 vs n",
                     Font14);
}

// "nlogn vs n" graph.

void GraphWindow::drawNLogNGraph(void)
{
  assert(XtIsRealized(_window));

  for(int i = 1;i < 50;i++)
  {
    _graph->drawLine(5 * i + 70,
                     280 - (int)(i * log(i) * 1.25),
                     5 * (i + 1) + 70,
                     280 - (int)((i + 1) * log(i + 1) * 1.25));
  }

  char str[5];
  for(i = 1;i <= 50;i+=10)
  {
    _graph->drawLine(65,
                     280 - (int)(i * log(i) * 1.25),
                     70,
                     280 - (int)(i * log(i) * 1.25));

    sprintf(str,"%.1f",i * log(double(i)));

    _graph->drawString(5,
                       280 - (int)(i * log(i) * 1.25),
```

```
                        str,
                        Font14);
  }

  _graph->drawString(100,
                     120,
                     "nlog(n) vs n",
                     Font14);
}

// Plot swaps.

void GraphWindow::plotSwaps(int swaps[],float fraction)
{
  assert(XtIsRealized(_window));

  for(int i = 0;i <= 5;i++)
  {
    _graph->drawCircle(70 + (i * 10 * 5),
                       280 - (int)(swaps[i] * fraction),
                       5);
  }
}

// Plot comparisons.

void GraphWindow::plotComps(int comps[],float fraction)
{
  assert(XtIsRealized(_window));

  for(int i = 0;i <= 5;i++)
  {
    _graph->drawRectangle(70 + (i * 10 * 5),
                          280 - (int)(comps[i] * fraction),
                          5,
                          5);
  }
}

// Copy canvas to the window upon expose.
void GraphWindow::expose(void)
{
  assert(_graph != NULL);

  _graph->copy();
}
////////////////////////////////////////////////////////////////////////////
// SpeedWindow.h : Support a window to get speed of display from the user.
////////////////////////////////////////////////////////////////////////////

#ifndef SPEEDWINDOW_H
#define SPEEDWINDOW_H

#include "BasicWindow.h"

class SpeedWindow:public BasicWindow
{
  private:

    int _speed;
    int _tempSpeed;

    Widget _okButton;       // Ok button to close window.
    Widget _speedScale,     // SpeedScale widget.

    // Callbacks.

    static void okCallback(Widget,XtPointer,XtPointer);
    static void cancelCallback(Widget,XtPointer,XtPointer);
    static void speedCallback(Widget,XtPointer,XtPointer);

    void okExecute(void);
    void cancelExecute(void);
    void speedExecute(int);

  public:
```

```
    SpeedWindow(void);
    ~SpeedWindow(void);

    int speed(void);

    virtual void createShell(void);
    virtual void createWidget(void);

    void manage(void);
    void reinit(void);
};

#endif
////////////////////////////////////////////////////////////////////////////
// SpeedWindow.C : Support a window to get speed of display from the user.
////////////////////////////////////////////////////////////////////////////

#include "SpeedWindow.h"
#include "Application.h"

#include <Xm/RowColumn.h>
#include <Xm/Scale.h>
#include <Xm/PushB.h>

// Constructor creates window, Scale widget, and pushbutton and registers
// callback functions.

SpeedWindow::SpeedWindow(void):BasicWindow("Change Speed","Speed")
{
  BasicWindow::initialize();

  XmString title;

  title = XmStringCreateLtoR("Frames Per Second",XmSTRING_DEFAULT_CHARSET);

  _speedScale = XtVaCreateWidget("Scale",
                                 xmScaleWidgetClass,
                                 _widget,
                                 XmNtitleString, title,
                                 XmNorientation, XmHORIZONTAL,
                                 XmNminimum,     1,
                                 XmNmaximum,     20,
                                 XmNvalue,       10,
                                 XmNshowValue,   TRUE,
                                 NULL);

  XtAddCallback(_speedScale,
                XmNvalueChangedCallback, &SpeedWindow::speedCallback,
                (XtPointer)this);

  _okButton = XtVaCreateWidget("              OK",
                               xmPushButtonWidgetClass,
                               _widget,
                               NULL);

  XtAddCallback(_okButton,
                XmNactivateCallback,&SpeedWindow::okCallback,
                (XtPointer)this);

  _speed = 10;
}

SpeedWindow::~SpeedWindow(void)
{
}

void SpeedWindow::createWidget(void)
{
  _widget = XtVaCreateWidget(_widgetName,
                             xmRowColumnWidgetClass,
                             _window,
                             NULL);
}

void SpeedWindow::createShell(void)
{
  _window = XtVaCreatePopupShell(_windowName,
```

```
                                topLevelShellWidgetClass,
                                theApplication->toplevel(),
                                XmNtitle,      _windowName,
                                XmNwidth,      200,
                                XmNminWidth,   200,
                                XmNmaxWidth,   200,
                                XmNheight,     100,
                                XmNminHeight,  100,
                                XmNmaxHeight,  100,
                                XmNx,          200,
                                XmNy,          100,
                                NULL);
}

void SpeedWindow::manage(void)
{
  BasicWindow::manage();

  XtManageChild(_speedScale);
  XtManageChild(_okButton);
}

void SpeedWindow::okCallback(Widget,XtPointer clientData,XtPointer)
{
  SpeedWindow* obj = (SpeedWindow*) clientData;

  obj->okExecute();
}

void SpeedWindow::okExecute(void)
{
  BasicWindow::unmanage();
}

void SpeedWindow::speedCallback(Widget,XtPointer clientData,XtPointer callData)
{
  XmScaleCallbackStruct* cb = (XmScaleCallbackStruct*) callData;
  SpeedWindow* obj          = (SpeedWindow*) clientData;

  obj->speedExecute(cb->value);
}

void SpeedWindow::speedExecute(int value)
{
  _speed = value;
}

int SpeedWindow::speed(void)
{
  return (1000 /_speed);
}
////////////////////////////////////////////////////////////////////////////
// StatsWindow.h : Support a window to display current statistics.
////////////////////////////////////////////////////////////////////////////

#ifndef STATSWINDOW_H
#define STATSWINDOW_H

#include "BasicWindow.h"

class StatsWindow:public BasicWindow
{
  private:

    Widget _okButton;
    Widget _statsLabel;

    static void okCallback(Widget,XtPointer,XtPointer);
    void        okExecute(void);

  public:

    StatsWindow(void);
    ~StatsWindow(void);

    virtual void createShell(void);
    virtual void createWidget(void);
```

```
    void manage(void);
    void reinit(int[],int[],int[]);
};

#endif
////////////////////////////////////////////////////////////////////////////
// StatsWindow.C : Support a window to display current statistics.
////////////////////////////////////////////////////////////////////////////

#include "StatsWindow.h"
#include "Application.h"
#include "Sort.h"
#include <Xm/Form.h>
#include <Xm/PushB.h>
#include <Xm/Label.h>
#include <stdio.h>
#include <stdlib.h>

StatsWindow::StatsWindow(void):BasicWindow("Statistics","Speed")
{
  BasicWindow::initialize();

  _statsLabel = XtVaCreateWidget("Stats",
                                 xmLabelWidgetClass, _widget,
                                 NULL);

  _okButton = XtVaCreateWidget("OK",
                               xmPushButtonWidgetClass,_widget,
                               NULL);

  XtAddCallback(_okButton,
                XmNactivateCallback,&StatsWindow::okCallback,
                (XtPointer)this);

  XtVaSetValues(_statsLabel,
                XmNtopAttachment,    XmATTACH_FORM,
                XmNleftAttachment,   XmATTACH_FORM,
                XmNrightAttachment,  XmATTACH_FORM,
                XmNbottomAttachment, XmATTACH_WIDGET,
                XmNbottomWidget,     _okButton,
                NULL);

  XtVaSetValues(_okButton,
                XmNbottomAttachment, XmATTACH_FORM,
                XmNleftAttachment,   XmATTACH_FORM,
                XmNrightAttachment,  XmATTACH_FORM,
                NULL);
}

StatsWindow::~StatsWindow(void)
{
}

void StatsWindow::createWidget(void)
{
  _widget = XtVaCreateWidget(_widgetName,
                             xmFormWidgetClass _window,
                             NULL);
}

void StatsWindow::createShell(void)
{
  _window = XtVaCreatePopupShell(_windowName,
                                 topLevelShellWidgetClass,
                                 theApplication->toplevel(),
                                 XmNtitle,      _windowName,
                                 XmNwidth,      400,
                                 XmNminWidth,   400,
                                 XmNmaxWidth,   400,
                                 XmNheight,     265,
                                 XmNminHeight,  265,
                                 XmNmaxHeight,  265,
                                 XmNx,          400,
                                 XmNy,          100,
                                 NULL);
}
```

```
void StatsWindow::reinit(int size[],int comps[],int swaps[])
{
  char     text[1000];
  char     temp[100];
  XmString str;

  strcpy(text,"   Sort Name    Size of Data  No. of Comparisons  No. of Swaps");
  strcat(text,"\n");
  strcat(text,"   ---------    ------------  ------------------  ------------");
  strcat(text,"\n\n");

  for(int i = 0;i < NoOfSorts;i++)
  {
    sprintf(temp,"%14s",sortName[i]);
    strcat(text,temp);

    if(size[i] > 0)
    {
      sprintf(temp," %8d  %16d  %14d\n\n",size[i],comps[i],swaps[i]);
    }
    else
    {
      sprintf(temp,"\n\n");
    }
    strcat(text,temp);
  }

  str = XmStringCreateLtoR(text,XmSTRING_DEFAULT_CHARSET);

  XtVaSetValues(_statsLabel,
                XmNlabelString, str,
                XmNalignment,   XmALIGNMENT_BEGINNINC,
                NULL);

  XmStringFree(str);
}

void StatsWindow::manage(void)
{
  BasicWindow::manage();

  XtManageChild(_okButton);
  XtManageChild(_statsLabel);
}

void StatsWindow::okCallback(Widget,XtPointer clientData,XtPointer)
{
  StatsWindow* obj = (StatsWindow*) clientData;

  obj->okExecute();
}

void StatsWindow::okExecute(void)
{
  BasicWindow::unmanage();
}
////////////////////////////////////////////////////////////////////////////////
// FSDialog.h : Support a file selection dialog box.
////////////////////////////////////////////////////////////////////////////////

#ifndef FSDIALOG_H
#define FSDIALOG_H

#include "BasicWidget.h"

typedef void(*FSCallback) (void*,char*);

class FSDialog:public BasicWidget
{
  private:

    // Static member callback functions.

    static void okCallback(Widget,XtPointer,XtPointer);
    static void cancelCallback(Widget,XtPointer,XtPointer);
    static void helpCallback(Widget,XtPointer,XtPointer);
```

```
  public:

    FSDialog(char*,FSCallback,void*);
    virtual ~FSDialog(void);

    // Function to be called when the user selects a file.

    FSCallback _callback;

    void* _clientData;          // Data provided by caller.

    void         fileSelected(char*);
    virtual void manage(void);
};

#endif
////////////////////////////////////////////////////////////////////////////
// FSDialog.C : Support a file selection dialog box.
////////////////////////////////////////////////////////////////////////////

#include "FSDialog.h"
#include "Application.h"
#include "PopupDialog.h"

#include <Xm/FileSB.h>

// Constructor creates the file selection dialog box and registers callback
// functions.

FSDialog::FSDialog(char* name,FSCallback callback,void* clientData):
                                                          BasicWidget(name)
{
  _callback   = callback;
  _clientData = clientData;

  _widget = XmCreateFileSelectionDialog(theApplication->toplevel(),name,NULL,0);

  XtAddCallback(_widget,
                XmNcancelCallback,&FSDialog::cancelCallback,
                (XtPointer)this);

  XtAddCallback(_widget,
                XmNokCallback,&FSDialog::okCallback,
                (XtPointer)this);

  XtAddCallback(_widget,
                XmNhelpCallback,&FSDialog::helpCallback,
                (XtPointer)this);
}

// Destroy the widget.

FSDialog::~FSDialog(void)
{
  XtDestroyWidget(_widget);
}

// This function is called when the user selects a file.

void FSDialog::okCallback(Widget,XtPointer clientData,XtPointer callData)
{
  FSDialog* obj = (FSDialog*)clientData;

  XmFileSelectionBoxCallbackStruct* cb =
                                  (XmFileSelectionBoxCallbackStruct*)callData;

  char*    fileName = NULL;
  XmString xmstr    = cb->value;
  int      status   = 0;

  if(xmstr)
  {
    status = XmStringGetLtoR(xmstr,XmSTRING_DEFAULT_CHARSET,&fileName);

    if(!status)
    {
    }
```

```
    else
    {
      obj->fileSelected(fileName);
    }
  }

  obj->BasicWidget::unmanage();
}

// Close the box without selecting a file.

void FSDialog::cancelCallback(Widget,XtPointer clientData,XtPointer)
{
  FSDialog* obj = (FSDialog*)clientData;

  obj->BasicWidget::unmanage();
}

// Help!

void FSDialog::helpCallback(Widget,XtPointer,XtPointer)
{
  PopupDialog* infoDialog = new PopupDialog("Help Info",INFO);
  infoDialog->postMessage("Select a data file");
}

void FSDialog::fileSelected(char* fileName)
{
  if(_callback)
  {
    _callback(_clientData,fileName);
  }
}

void FSDialog::manage(void)
{
  BasicWidget::manage();
  XtPopup(XtParent(_widget),XtGrabNone);
  XMapRaised(theApplication->display(),XtWindow(XtParent(_widget)));
}
////////////////////////////////////////////////////////////////////////////////
// OpenCmd.h : Support a pushbutton to get user input data file.
////////////////////////////////////////////////////////////////////////////////

#ifndef OPENCMD_H
#define OPENCMD_H

#include "CmdButton.h"
#include "UnsortedData.h"

// Forward references.

class FSDialog;
class DisplayWindow;
class CmdList;

class OpenCmd:public CmdButton
{
  private:

    FSDialog*      _fsDialog;      // File selection dialog box.
    DisplayWindow* _displayWindow; // Display window to display unsorted data.

    UnsortedData _data;            // User input data.
    CmdList*     _activationList;  // Commands to be activated.

    // Callback function.

    static void readFileCallback(void*,char*);
    void        readData(char*);

  public:

    OpenCmd(Widget,int);
    ~OpenCmd(void);

    // Add to commands activation list.
```

```
    void addToActivationList(CmdInterface*);

    // Execute callback.

    void execute(Widget,XtPointer);

    // Return input data.

    UnsortedData data(void);
};

#endif
////////////////////////////////////////////////////////////////////////////
// OpenCmd.C : Support a pushbutton to get user input data.
////////////////////////////////////////////////////////////////////////////

#include "OpenCmd.h"
#include "CmdList.h"
#include "FSDialog.h"
#include "PopupDialog.h"
#include "DisplayWindow.h"

#include <fstream.h>
#include <ctype.h>

// Constructor creates a file selection box and a display window.
// Also initializes a command activation list.

OpenCmd::OpenCmd(Widget parent,int active):CmdButton("Open",parent,active)
{
  _displayWindow   = new DisplayWindow("Unsorted Data");

  _fsDialog        = new FSDialog("File Selection",
                                  &OpenCmd::readFileCallback,(void*)this);

  _activationList  = new CmdList("Open: Activation List");
}

OpenCmd::~OpenCmd(void)
{
  delete _fsDialog;
  delete _displayWindow;
  delete _activationList;
}

void OpenCmd::execute(Widget,XtPointer)
{
  _fsDialog->manage();
}

void OpenCmd::readFileCallback(void* clientData,char* fileName)
{
  OpenCmd* obj = (OpenCmd*)clientData;

  obj->readData(fileName);
}

// Read input file and store the data. If incorrect file or invalid data popup
//an error or warning dialog

void OpenCmd::readData(char* fileName)
{
  int  data[50];
  int  size = 0;
  char c;

  ifstream inFile(fileName,ios::in);

  if(!inFile)
  {
    theErrorDialog->postMessage("Invalid file!\nCannot open file.",NULL,TRUE);
    return;
  }

  inFile >> c;
  if(inFile.peek() != EOF)
  {
```

```
    if(!isdigit(c))
    {
      theErrorDialog->postMessage("Invalid file!\nSelect a data file.",
                                                                    NULL,TRUE);
      return;
    }
    inFile.putback(c);
  }

  while(inFile && (size < 50))
  {
    inFile >> data[size];

    if(data[size] > 99)
    {
      data[size] = 99;
    }
    size++;

    inFile >> c;


    if(inFile.peek() != EOF)
    {
      if(!isdigit(c))
      {
        theErrorDialog->postMessage("Invalid File!\nSelect a data file.",
                                                                    NULL,TRUE);
        return;
      }

      inFile.putback(c);
    }
  }

  if(size == 0)
  {
    theErrorDialog->postMessage("Empty File!\n",NULL,TRUE);

    return;
  }

  if(size == 50)
  {
    theWarnDialog->postMessage("Only the first 50 numbers are sorted.",
                                                                    NULL,TRUE);
  }

  inFile.close();

  _activationList->activate();

  _data.copy(data,size);

  _displayWindow->reinit(_data);
  _displayWindow->draw();
  _displayWindow->nextFrame();
}

UnsortedData OpenCmd::data(void)
{
  return _data;
}

void OpenCmd::addToActivationList(CmdInterface* cmd)
{
  _activationList->add(cmd);
}
////////////////////////////////////////////////////////////////////////////
// ExitCmd.h : Support a pushbutton.
//             Pressing the "Exit" button ends the program.
////////////////////////////////////////////////////////////////////////////

#ifndef EXITCMD_H
#define EXITCMD_H

#include "CmdButton.h"
```

```
class ExitCmd:public CmdButton
{
  public:

    ExitCmd(Widget parent,int active):CmdButton("Exit",parent,active){}
    ~ExitCmd(void){}

    void  execute(Widget,XtPointer);
};

#endif
////////////////////////////////////////////////////////////////////////////
// ExitCmd.C : Support a pushbutton.
//             Pressing the Exit button ends the program.
////////////////////////////////////////////////////////////////////////////

#include "ExitCmd.h"
#include "Application.h"

#include <stdlib.h>

void ExitCmd::execute(Widget,XtPointer)
{
  delete theApplication;

  exit(0);
}
////////////////////////////////////////////////////////////////////////////
// ResetCmd.h : Support a pushbutton to reset the display windows.
////////////////////////////////////////////////////////////////////////////

#ifndef RESETCMD_H
#define RESETCMD_H

#include "CmdButton.h"
#include "SortList.h"

class ResetCmd:public CmdButton
{
  private:

    SortList*  _sortList[NoOfSorts]; // List of sorting methods (togglebuttons).

  public:

    ResetCmd(Widget,int);
    ~ResetCmd(void);

    // Add to the list.

    void    add(SortList**);

    // Execute callback.

    virtual void execute(Widget,XtPointer);
};

#endif
////////////////////////////////////////////////////////////////////////////
// ResetCmd.C : Support a pushbutton to reset display windows.
////////////////////////////////////////////////////////////////////////////

#include "ResetCmd.h"

// Constructor creates the pushbutton.

ResetCmd::ResetCmd(Widget parent,int active):CmdButton("Reset",parent,active)
{
}

ResetCmd::~ResetCmd(void)
{
}

void ResetCmd::add(SortList** list)
{
  for(int i = 0;i < NoOfSorts;i++)
```

```
  {
    _sortList[i] = list[i];
  }
}

void ResetCmd::execute(Widget,XtPointer)
{
  for(int i = 0;i < NoOfSorts;i++)
  {
    if(_sortList[i]->isSelected())
    {
      _sortList[i]->reinit();
    }
  }
}
////////////////////////////////////////////////////////////////////////////////
// SortCmd.h : Support a pushbutton to start display in display windows.
////////////////////////////////////////////////////////////////////////////////

#ifndef SORTCMD_H
#define SORTCMD_H

#include "CmdButton.h"
#include "SortList.h"

class SortCmd:public CmdButton
{
  private:

    SortList*  _sortList[NoOfSorts]; // List of sorting methods (togglebuttons).

  public:

    SortCmd(Widget,int);
    ~SortCmd(void);

    // Add to the list.

    void add(SortList**);

    // Execute callback.

    virtual void execute(Widget,XtPointer);
};

#endif
////////////////////////////////////////////////////////////////////////////////
// SortCmd.C : Support a pushbutton to start display in display windows.
////////////////////////////////////////////////////////////////////////////////

#include "SortCmd.h"

// Constructor creates the pushbutton.

SortCmd::SortCmd(Widget parent,int active):
                                    CmdButton("Display Sorting",parent,active)
{
}

SortCmd::~SortCmd(void)
{
}

void SortCmd::add(SortList** list)
{
  for(int i = 0;i < NoOfSorts;i++)
  {
    _sortList[i] = list[i];
  }
}

// Check all sorting togglebuttons and start display for those that have been
// selected.

void SortCmd::execute(Widget,XtPointer)
{
  for(int i = 0;i < NoOfSorts;i++)
```

```
  {
    if(_sortList[i]->isSelected())
    {
      _sortList[i]->displaySort();
    }
  }
}
////////////////////////////////////////////////////////////////////////////
// SpeedCmd.h : Support a pushbutton which would popup a SpeedWindow.
////////////////////////////////////////////////////////////////////////////

#ifndef SPEEDCMD_H
#define SPEEDCMD_H

#include "CmdButton.h"

// Forward reference.

class SpeedWindow;

class SpeedCmd:public CmdButton
{
  private:

    SpeedWindow* _speedWindow;          // SpeedWindow object.

  public:

    SpeedCmd(Widget parent,int active);
    virtual ~SpeedCmd(void);

    // Execute callback.

    virtual void execute(Widget,XtPointer);

    // Return current speed.

    int speed(void);
};

#endif
////////////////////////////////////////////////////////////////////////////
// SpeedCmd.h : Support a pushbutton which would popup a SpeedWindow.
////////////////////////////////////////////////////////////////////////////

#include "SpeedCmd.h"
#include "SpeedWindow.h"

// Constructor creates the pushbutton and initializes SpeedWindow object.

SpeedCmd::SpeedCmd(Widget parent,int active):CmdButton("Speed",parent,active)
{
  _speedWindow = new SpeedWindow();
}

SpeedCmd::~SpeedCmd(void)
{
  delete _speedWindow;
}

void SpeedCmd::execute(Widget,XtPointer)
{
  _speedWindow->manage();
}

int SpeedCmd::speed(void)
{
  return _speedWindow->speed();
}
////////////////////////////////////////////////////////////////////////////
// StatsCmd.h : Support a pushbutton to popup StatsWindow.
////////////////////////////////////////////////////////////////////////////

#ifndef STATSCMD_H
#define STATSCMD_H

#include "CmdButton.h"
```

```
#include "SortList.h"

// Forward reference.

class StatsWindow;

class StatsCmd:public CmdButton
{
  private:

    SortList*     _sortList[NoOfSorts];
    StatsWindow* _statsWindow;

  public:

    StatsCmd(Widget,int);
    virtual ~StatsCmd(void);

    void          add(SortList**);
    virtual void execute(Widget,XtPointer);
};

#endif
////////////////////////////////////////////////////////////////////////////////
// StatsCmd.C : Support a pushbutton to popup a StatsWindow.
////////////////////////////////////////////////////////////////////////////////

#include "StatsCmd.h"
#include "StatsWindow.h"

StatsCmd::StatsCmd(Widget parent,int active):
                                              CmdButton("Statistics",parent,active)
{
  _statsWindow = new StatsWindow();
}

StatsCmd::~StatsCmd(void)
{
}

void StatsCmd::add(SortList** list)
{
  for(int i = 0;i < NoOfSorts;i++)
  {
    _sortList[i] = list[i];
  }
}

void StatsCmd::execute(Widget,XtPointer)
{
  int size[NoOfSorts];
  int comps[NoOfSorts];
  int swaps[NoOfSorts];

  for(int i = 0;i < NoOfSorts;i++)
  {
    size[i]  = _sortList[i]->size();
    comps[i] = _sortList[i]->comps();
    swaps[i] = _sortList[i]->swaps();
  }

  _statsWindow->reinit(size,comps,swaps);
  _statsWindow->manage();
}
////////////////////////////////////////////////////////////////////////////////
// CmdList.h : Support a list of command interface objects.
////////////////////////////////////////////////////////////////////////////////

#ifndef CMDLIST_H
#define CMDLIST_H

#include "CmdInterface.h"

typedef CmdInterface* Command;

class CmdList:public CmdInterface
{
```

```
  private:

    Command* _listOfCmds;           // List of CmdInterface objects.
    int      _size;                 // Size of the list.

  public:

    CmdList(const char*);
    ~CmdList(void);

    Command operator[](int);        // Overloaded operator.
    int     size(void);             // Return size of list.
    void    add(Command);           // Add a pointer to CmdInterface.

    // Activate and deactivate the list.

    virtual void activate(void);
    virtual void deactivate(void);

    // Manage and unmangage the list.

    virtual void manage(void);
    virtual void unmanage(void);

    // Execute all the CmdInterface objects in the list.

    virtual void execute(Widget,XtPointer){}
};

#endif
////////////////////////////////////////////////////////////////////////////
// CmdList.C : Support a list of command interface objects.
////////////////////////////////////////////////////////////////////////////

#include "CmdList.h"

#include <X11/Intrinsic.h>
#include <assert.h>

// Constructor initializes the size and the list of commands to zero.

CmdList::CmdList(const char* name = "List"):CmdInterface(name)
{
  _size       = 0;
  _listOfCmds = new Command[100];
}

CmdList::~CmdList(void)
{
  delete[] _listOfCmds;
}

Command CmdList::operator[](int i)
{
  assert((i >= 0) || (i < _size));
  return _listOfCmds[i];
}

int CmdList::size(void)
{
  return _size;
}

void CmdList::add(Command cmd)
{
  _listOfCmds[_size++] = cmd;
}

void CmdList::activate(void)
{
  for(int i = 0;i < _size;i++)
  {
    _listOfCmds[i]->activate();
  }
}

void CmdList::deactivate(void)
```

```
{
  for(int i = 0;i < _size;i++)
  {
    _listOfCmds[i]->deactivate();
  }
}

void CmdList::manage(void)
{
  for(int i = 0;i < _size;i++)
  {
    _listOfCmds[i]->manage();
  }
}

void CmdList::unmanage(void)
{
  for(int i = 0;i < _size;i++)
  {
    _listOfCmds[i]->unmanage();
  }
}
////////////////////////////////////////////////////////////////////////////////
// SortList.h : Support a list of togglebuttons for the user to check the
//              sorting methods to be used.
////////////////////////////////////////////////////////////////////////////////

#ifndef SORTLIST_H
#define SORTLIST_H

#include "CmdToggle.h"
#include "Sort.h"
#include "UnsortedData.h"

// Forward references.

class OpenCmd;
class SpeedCmd;
class SortInfo;
class DisplayWindow;

class SortList:public CmdToggle
{
  private:

    DisplayWindow* _displayWindow; // Display window object.

    OpenCmd*  _open;               // OpenCmd has the unsorted data.
    SpeedCmd* _speed;              // SpeedCmd has the current speed of display.

    Sort*        _sort;            // Sort is the abtract class for all sorts.
    SortInfo*    _info;            // Sorting info is stored in SortInfo.
    UnsortedData _data;            // Unsorted data is stored in UnsortedData.
    int          _time;            // To time the display.
    XtIntervalId _id;
    int          _cnt;             // Count the number of sorting steps.

    int _displayOff;               // Boolean value to check if the display
                                   // window is already active.

    // Callbacks.

    static void timerCallback(XtPointer,XtIntervalId*);
    void   timeTheDisplay(void);

  public:

    SortList(const char*,int,Widget,int);
    ~SortList(void);

    // Execute callback.

    virtual void execute(Widget,XtPointer);

    // Uncheck togglebutton and close window.

    virtual void deSelect(void);
```

```
    // Controls. OpenCmd to get unsorted data. SpeedCmd to get speed of display.

    void setControls(OpenCmd*,SpeedCmd*);

    // Reinitialize display with current data.

    void reinit(void);

    // Start display.

    void displaySort(void);

    int size(void);
    int swaps(void);
    int comps(void);
};

#endif
////////////////////////////////////////////////////////////////////////////////
// SortList.C :
////////////////////////////////////////////////////////////////////////////////

#include "SortList.h"
#include "Application.h"
#include "DisplayWindow.h"
#include "OpenCmd.h"
#include "SpeedCmd.h"
#include "SortInfo.h"
#include "SelectionSort.h"
#include "BubbleSort.h"
#include "CombSort.h"
#include "InsertionSort.h"
#include "QuickSort.h"
#include "ShellSort.h"

SortList::SortList(const char* sortName,int sortClass,Widget parent,int active):
                                                    CmdToggle(sortName,parent,active)
{
  _displayWindow = new DisplayWindow(sortName,this);

  switch(sortClass)
  {
    case InsertionSortClass:

      _sort = new InsertionSort();
      break;

    case SelectionSortClass:

      _sort = new SelectionSort();
      break;

    case QuickSortClass:

      _sort = new QuickSort();
      break;

    case ShellSortClass:

      _sort = new ShellSort();
      break;

    case BubbleSortClass:

      _sort = new BubbleSort();
      break;

    case CombSortClass:

      _sort = new CombSort();
      break;

    default:
      break;
  }

  _displayOff = TRUE;
```

```
  _id = NULL;
}

SortList::~SortList(void)
{
  delete _displayWindow;
}

void SortList::reinit(void)
{
  _displayWindow->reinit(_open->data());
  _displayWindow->draw();
  _displayWindow->nextFrame();

  if(_id)
  {
    XtRemoveTimeOut(_id);
  }

  _displayOff = TRUE;

  _cnt = 0;
}

void SortList::execute(Widget,XtPointer)
{
  if(CmdToggle::isSelected())
  {
    reinit();
  }

  else
  {
    _displayWindow->ok();
  }
}

void SortList::deSelect(void)
{
  CmdToggle::deSelect();

  _displayOff = TRUE;
}

void SortList::setControls(OpenCmd* open,SpeedCmd* speed)
{
  _open  = open;
  _speed = speed;
}

void SortList::displaySort(void)
{
  if(_displayOff)
  {
    _displayOff = FALSE;

    _sort->reinit();

    _info = _sort->doTheSorting(_open->data());

    _id   = XtAppAddTimeOut(theApplication->appContext(),_speed->speed(),
                                        &SortList::timerCallback,(XtPointer)this);
  }
}

void SortList::timerCallback(XtPointer clientData,XtIntervalId*)
{
  SortList* obj = (SortList*)clientData;
  obj->timeTheDisplay();
}

void SortList::timeTheDisplay(void)
{
  if(!CmdToggle::isSelected())
  {
    if(_id)
    {
```

```
      XtRemoveTimeOut(_id);
    }

    return;
  }

  if(_cnt < _sort->steps())
  {
    _displayWindow->draw(_info[_cnt++]);
    _displayWindow->nextFrame();

    if(_id)
    {
      XtRemoveTimeOut(_id);
    }

    _id = XtAppAddTimeOut(theApplication->appContext(),_speed->speed(),
                          &SortList::timerCallback,(XtPointer)this);
  }

  else
  {
    _displayWindow->draw();
    _displayWindow->nextFrame();

    if(_id)
    {
      XtRemoveTimeOut(_id);
      _displayOff = TRUE;
    }
  }
}

int SortList::size(void)
{
  return _sort->size();
}

int SortList::comps(void)
{
  return _sort->comps();
}

int SortList::swaps(void)
{
  return _sort->swaps();
}
////////////////////////////////////////////////////////////////////////////
// GraphList.h : Support a list of pushbuttons, executing which would display
//               the corresponding graph for the sorting algorithm.
////////////////////////////////////////////////////////////////////////////

#ifndef GRAPHLIST_H
#define GRAPHLIST_H

#include "CmdButton.h"

// Forward reference.

class GraphWindow;

class GraphList:public CmdButton
{
  private:

    GraphWindow* _graphWindow;              // GraphWindow object.
    int          _sortClass;                // Sorting class (algorithm).

  public:

    GraphList(const char*,int,Widget,int);
    ~GraphList(void);

    virtual void execute(Widget,XtPointer);
};

#endif
```

```
//////////////////////////////////////////////////////////////////////////////
// GraphList.C : Support a list of pushbuttons, executing which would display
//               the corresponding graph for the sorting algorithm.
//////////////////////////////////////////////////////////////////////////////

#include "GraphList.h"
#include "GraphWindow.h"
#include "Sort.h"

// Constructor creates a GraphWindow object, which would display the graph.

GraphList::GraphList(const char* sortName,int sortClass,
                     Widget parent,int active):CmdButton(sortName,parent,active)
{
  _graphWindow = new GraphWindow(sortName);
  _sortClass   = sortClass;
}

GraphList::~GraphList(void)
{
  delete _graphWindow;
}

// Corresponding graph is drawn based on experimental data.

void GraphList::execute(Widget,XtPointer)
{
  static int insertSwaps[6] = {0,10,19,29,39,49};
  static int insertComps[6] = {0,18,104,406,708,1110};

  static int selectSwaps[6] = {0,11,20,30,40,50};
  static int selectComps[6] = {0,55,190,435,780,1225};

  static int bubbleSwaps[6] = {0,29,64,96,128,250};
  static int bubbleComps[6] = {0,66,210,465,820,1200};

  static int combSwaps[6]   = {0,10,20,30,40,361};
  static int combComps[6]   = {0,52,99,121,151,297};

  static int shellSwaps[6]  = {0,22,60,90,120,200};
  static int shellComps[6]  = {0,16,28,142,156,270};

  static int quickSwaps[6]  = {0,12,22,33,44,55};
  static int quickComps[6]  = {0,29,71,97,103,129};

  _graphWindow->reinit();

  switch(_sortClass)
  {
    case InsertionSortClass:

      _graphWindow->drawNSquareGraph();
      _graphWindow->plotSwaps(insertSwaps,0.10);
      _graphWindow->plotComps(insertComps,0.10);

      break;

    case SelectionSortClass:

      _graphWindow->drawNSquareGraph();
      _graphWindow->plotSwaps(selectSwaps,0.10);
      _graphWindow->plotComps(selectComps,0.10);

      break;

    case QuickSortClass:

      _graphWindow->drawNLogNGraph();
      _graphWindow->plotSwaps(quickSwaps,1.25);
      _graphWindow->plotComps(quickComps,1.25);

      break;

    case ShellSortClass:

      _graphWindow->drawNSquareGraph();
      _graphWindow->plotSwaps(shellSwaps,0.10);
```

```
      _graphWindow->plotComps(shellComps,0.10);

      break;

    case BubbleSortClass:

      _graphWindow->drawNSquareGraph();
      _graphWindow->plotSwaps(bubbleSwaps,0.10);
      _graphWindow->plotComps(bubbleComps,0.10);

      break;

    case CombSortClass:

      _graphWindow->drawNLogNGraph();
      _graphWindow->plotSwaps(combSwaps,1.25);
      _graphWindow->plotComps(combComps,1.25);

      break;
  }
}
////////////////////////////////////////////////////////////////////////////////
// InfoList.h : Support "Info" menu.
//              Provide information on Sorting algorithms implemented.
////////////////////////////////////////////////////////////////////////////////

#ifndef INFOLIST_H
#define INFOLIST_H

#include "CmdButton.h"

class InfoList:public CmdButton
{
  private:

    int _sortClass;

  public:

    InfoList(const char*,int,Widget,int);
    ~InfoList(void);

    virtual void execute(Widget,XtPointer);
};

#endif
////////////////////////////////////////////////////////////////////////////////
// InfoList.C : Support "Info" menu.
//              Provide information the sorting algorithms implemented.
////////////////////////////////////////////////////////////////////////////////

#include "InfoList.h"
#include "PopupDialog.h"

#include "Sort.h"
#include <string.h>

// Constructor creates a pushbutton for the menu item.

InfoList::InfoList(const char* sortName,int sortClass,Widget parent,int active):
                                                    CmdButton(sortName,parent,active)
{
  _sortClass = sortClass;
}

InfoList::~InfoList(void)
{
}

// Popup a dialog with information.

void InfoList::execute(Widget,XtPointer)
{
  char temp1[1000];
  char temp2[1000];

  switch(_sortClass)
```

```
{
  case InsertionSortClass:

    strcpy(temp1,"Insertion Sort\n");
    strcat(temp1,"This algorithm could be compared with the task of\n");
    strcat(temp1,"picking up cards for a bridge hand. Each new card\n");
    strcat(temp1,"is inserted into the correct position relative to\n");
    strcat(temp1,"the other cards already in the hand.");

    strcpy(temp2,"Performance Analysis of Insertion Sort\n\n");
    strcat(temp2,"The sort uses about n^2/2 comparisons and n^2/8\n");
    strcat(temp2,"swaps on the average and twice as many in the worst\n");
    strcat(temp2,"case to sort n records. So the algorithm has a running\n");
    strcat(temp2,"time complexity of the order of n^2. If the input is\n");
    strcat(temp2,"pre-sorted, running time would be of the order of n.\n");
    strcat(temp2,"Since sorting is done in-place, no extra memory is \n");
    strcat(temp2,"needed. So the space complexity of the algorithm is \n");
    strcat(temp2,"of the order of n.");

    break;

  case ShellSortClass:

    strcpy(temp1,"Shell Sort\n");
    strcat(temp1,"It was one of the first algorithms to break the\n");
    strcat(temp1,"quadratic time barrier. It works by making comparisons \n");
    strcat(temp1,"between elements that are distant;the distance between\n");
    strcat(temp1,"comparisons decreases as the algorithm runs, and in the\n");
    strcat(temp1,"last phase, adjacent elements are compared. For this\n");
    strcat(temp1,"reason, Shellsort is sometimes referred to as \n");
    strcat(temp1,"diminishing increment sort.");

    strcpy(temp2,"Performance Analysis of Shell Sort\n\n");
    strcpy(temp2,"The worst case running time complexity is of the\n");
    strcat(temp2,"order of n^2. But on the average the sort is much\n");
    strcat(temp2,"faster, though. Sorting is done in-place and so no\n");
    strcat(temp2,"extra space is required.");

    break;

  case BubbleSortClass:

    strcpy(temp1,"Bubble Sort\n");
    strcat(temp1,"The method is called \"bubble sorting\" because large\n");
    strcat(temp1,"elements \"bubble up\" to their proper position.\n");
    strcat(temp1,"Bubblesort is also known as exchange selection or\n");
    strcat(temp1,"propagation.");

    strcpy(temp2,"Performance Analysis of Bubble Sort\n\n");
    strcat(temp2,"Bubble Sort uses about n^2/2 comparisons and n^2\n");
    strcat(temp2,"exchanges on the average and in the worst case. If\n");
    strcat(temp2,"the input is pre-sorted no swaps are ever needed\n");
    strcat(temp2,"and only n comparisons are made. So the running time\n");
    strcat(temp2,"complexity would be of the order of n. Sorting is\n");
    strcat(temp2,"done in-place and so no extra memory is required. So\n");
    strcat(temp2,"the space complexity is of the order of n.\n");

    break;

  case CombSortClass:

    strcpy(temp1,"Comb Sort\n");
    strcat(temp1,"Bubblesort is modified to eliminate turtles by\n");
    strcat(temp1,"allowing the gap between elements that are compared\n");
    strcat(temp1,"to be greater than 1. This minor change makes the\n");
    strcat(temp1,"Combsort efficient while still retaining the\n");
    strcat(temp1,"simplicity of bubblesort.");

    strcpy(temp2,"Performance Analysis of Comb Sort\n\n");
    strcat(temp2,"Empirical results show that the average case running\n");
    strcat(temp2,"time complexity to be comparable to that of Quicksort.\n");
    strcat(temp2,"So the running time complexity appears to be of the\n");
    strcat(temp2,"order of nlog(n) for n records to be sorted. Sorting\n");
    strcat(temp2,"is done in-place and so no extra memory is required.\n");

    break;
```

```
    case QuickSortClass:

      strcpy(temp1,"Quick Sort\n");
      strcat(temp1,"As the name implies, this is the fastest known\n");
      strcat(temp1,"sorting algorithm in practice. This is a divide\n");
      strcat(temp1,"and conquer recursive algorithm.");

      strcpy(temp2,"Performance Analysis of Quick Sort\n\n");
      strcat(temp2,"The average case running time is of the order of\n");
      strcat(temp2,"nlog(n) and the worst case time complexity is of\n");
      strcat(temp2,"the order of n^2 for n records to be sorted.\n");
      strcat(temp2,"Sorting is done in-place and in the average case\n");
      strcat(temp2,"uses only a small auxillary stack to handle\n");
      strcat(temp2,"However, in the worst case, the extra space\n");
      strcat(temp2,"required to handle recursion will be about the size\n");
      strcat(temp2,"of the number of records to be sorted.");

      break;

    case SelectionSortClass:

      strcpy(temp1,"Selection Sort\n");
      strcat(temp1,"This is probably one of the simplest algorithms.\n");
      strcat(temp1,"While sorting, an item is moved to the beginning\n");
      strcat(temp1,"of the array by exchanging it with the item in the\n");
      strcat(temp1,"desired location.\n");

      strcpy(temp2,"Performance Analysis of Selection Sort\n\n");
      strcat(temp2,"The sort uses about n^2 comparisons and n exchanges\n");
      strcat(temp2,"to sort an array on n elements. The worst case and\n");
      strcat(temp2,"average case time complexity is of the order of n^2\n");
      strcat(temp2,"Since sorting is done in-place, no extra memory is\n");
      strcat(temp2,"needed. So the space complexity of the algorithm is\n");
      strcat(temp2,"of the order of n.");

      break;

    default:
    break;
  }

  theInfoDialog->postMessage(temp1,temp2);
}
////////////////////////////////////////////////////////////////////////////////
// HelpList.h : Support "Help" menu.
////////////////////////////////////////////////////////////////////////////////

#ifndef HELPLIST_H
#define HELPLIST_H

#include "CmdButton.h"

// Define help classes.

const int HelpClass1 = 1;
const int HelpClass2 = 2;
const int HelpClass3 = 3;
const int HelpClass4 = 4;
const int HelpClass5 = 5;

const int NoOfHelps  = 5;

extern char* const helpName[NoOfHelps];
extern int   const helpClass[NoOfHelps];

class HelpList:public CmdButton
{
  private:

    int _sortClass;              // Help class.

  public:

    HelpList(const char*,int,Widget,int);
    ~HelpList(void);

    virtual void execute(Widget,XtPointer);
```

```
};

#endif
////////////////////////////////////////////////////////////////////////////////
// HelpList.C : Support "Help" menu.
////////////////////////////////////////////////////////////////////////////////

#include "HelpList.h"
#include "PopupDialog.h"
#include "Sort.h"
#include <string.h>

// Define help menu.

char* const helpName[NoOfHelps]   = {"Data",
                                     "Sorting",
                                     "Display",
                                     "Statistics",
                                     "Graphs"};

int   const helpClass[NoOfHelps] = {1,2,3,4,5};

// Constructor creates a pushbutton for the help menu item.

HelpList::HelpList(const char* sortName,int sortClass,Widget parent,int active):
                                                  CmdButton(sortName,parent,active)
{
  _sortClass = sortClass;
}

HelpList::~HelpList(void)
{
}

// Popup a dialog with information to display.

void HelpList::execute(Widget,XtPointer)
{
  char temp1[1000];
  char temp2[1000];

  switch(_sortClass)
  {
    case HelpClass1:

      strcpy(temp1,"Data file contains unsorted data to be sorted. The\n");
      strcat(temp1,"input data has to be only positive integers and any\n");
      strcat(temp1,"other character is not recognized and results in an\n");
      strcat(temp1,"error.");

      strcpy(temp2,"For a good visual appeal, the maximum value of number\n");
      strcat(temp2,"that can be used in the unsorted data is limited to \n");
      strcat(temp2,"99 (any number more than 99 is truncated to 99). And\n");
      strcat(temp2,"also the maximum size of the data that can be sorted \n");
      strcat(temp2,"is 50 (if data size is more than 50, only the first 50\n");
      strcat(temp2,"numbers are sorted).");

      theInfoDialog->postMessage(temp1,temp2);

      break;

    case HelpClass2:

      strcpy(temp1,"Sorting as implemented here is the process of\n");
      strcat(temp1,"arranging a sequence of non-negative integers in\n");
      strcat(temp1,"an increasing order.");

      theInfoDialog->postMessage(temp1);

      break;

    case HelpClass3:

      strcpy(temp1,"The data is represented by rectangles whose length is\n");
      strcat(temp1,"sized according to the value they represent. The\n");
      strcat(temp1,"dynamic process of sorting is shown by darkening the\n");
      strcat(temp1,"rectangles that represent the numbers that are being\n");
```

```
      strcat(temp1,"compared or swapped at each step of sorting.\n");

      theInfoDialog->postMessage(temp1);

      break;

    case HelpClass4:

      strcpy(temp1,"The statistics shown are the statistics for the last\n");
      strcat(temp1,"run for each of the sorting method.");

      theInfoDialog->postMessage(temp1);

      break;

    case HelpClass5:

      strcpy(temp1,"The graphs provide complexity and performance analysis\n");
      strcat(temp1,"for the sorting methods. The graphs plot the number of\n");
      strcat(temp1,"comparisons and swaps that each sorting method took\n");
      strcat(temp1,"to sort some samples of data. These graphs also plot\n");
      strcat(temp1,"the average case complexity graphs.");

      theInfoDialog->postMessage(temp1);

      break;

    default:
    break;
  }
}
////////////////////////////////////////////////////////////////////////////
// AboutList.h : Support "About" menu.
////////////////////////////////////////////////////////////////////////////

#ifndef ABOUTLIST_H
#define ABOUTLIST_H

#include "CmdButton.h"

const int AboutClass1 = 1;
const int AboutClass2 = 2;

const int NoOfAbouts = 2;

extern char* const aboutName[NoOfAbouts];
extern int   const aboutClass[NoOfAbouts];

class AboutList:public CmdButton
{
  private:

    int _aboutClass;

  public:

    AboutList(const char*,int,Widget,int);
    ~AboutList(void);

    // Execute callback.

    virtual void execute(Widget,XtPointer);
};

#endif
////////////////////////////////////////////////////////////////////////////
// AboutList.C : Support "About" menu.
////////////////////////////////////////////////////////////////////////////

#include "AboutList.h"
#include "PopupDialog.h"
#include <string.h>

// Menu item names in "About" menu.

char* const aboutName[NoOfAbouts]   = {"SortDisplay",
                                       "Running SortDisplay"};
```

```
int   const aboutClass[NoOfAbouts] = {1,2};

AboutList::AboutList(const char* name,int aboutClass,Widget parent,int active):
                                                    CmdButton(name,parent,active)
{
  _aboutClass = aboutClass;
}

AboutList::~AboutList(void)
{
}

void AboutList::execute(Widget,XtPointer)
{
  char temp1[1000];
  char temp2[1000];

  switch(_aboutClass)
  {
    case AboutClass1:

      strcpy(temp1,"Type:     Visualization of Sorting Algorithms\n");
      strcat(temp1,"Authors:  Vikas Muktavaram & Dr. M.H. Samadzadeh\n");
      strcat(temp1,"Language: C++\n");
      strcat(temp1,"Toolkits: Motif, Xt Intrinsics and Xlib\n");
      strcat(temp1,"Date:     30 June 1996");

      strcpy(temp2,"This program is a visual tool which could help users\n");
      strcat(temp2,"get a grasp of sorting algorithms. The program gives\n");
      strcat(temp2,"a dynamic display of sorting methods while in execution\n");
      strcat(temp2,"and also provides some useful information about the\n");
      strcat(temp2,"algorithms.");

      break;

    case AboutClass2:

      strcpy(temp1,"Steps to run the program:\n");
      strcat(temp1,"1. Use \"Open\" under \"File\" menu to open a data\n");
      strcat(temp1,"   file.\n");
      strcat(temp1,"2. Select any number of the 6 sorting methods under\n");
      strcat(temp1,"   \"Sort\" menu.\n");
      strcat(temp1,"3. Use \"Display Sorting\" under \"Sort\" menu to start\n");
      strcat(temp1,"   all the sorts selected.\n");

      strcpy(temp2,"4. \"Reset\" command under \"Sort\" menu can be used to\n");
      strcat(temp2,"   restart sorting at any time.\n");
      strcat(temp2,"5. \"Speed\" command under \"Options\" menu can be used\n");
      strcat(temp2,"   to change the speed of display\n");
      strcat(temp2,"6. Other commands under \"Info\" menu provide useful\n");
      strcat(temp2,"   information about the sorting algorithms.");

      break;

    default:
    break;
  }

  theInfoDialog->postMessage(temp1,temp2);
}
////////////////////////////////////////////////////////////////////////////
// PopupDialog.h : Dialog manager class.
//                 Declare three types of dialogs as external objects.
//                 Create more dialogs if needed and destroy dialogs created
//                 temporarily when no longer needed.
////////////////////////////////////////////////////////////////////////////

#ifndef POPUPDIALOG_H
#define POPUPDIALOG_H

#include "BasicWidget.h"

const int INFO  = 0;
const int ERROR = 1;
const int WARN  = 2;

class PopupDialog:public BasicWidget
```

```
{
  private:

    int    _class;             // Dialog class.

    // Callbacks.

    static void okCallback(Widget,XtPointer,XtPointer);
    static void moreCallback(Widget,XtPointer,XtPointer);
    void        moreExecute(Widget);

    Widget getDialog(void);
    Widget createDialog(Widget);

  public:

    PopupDialog(char*,int);
    ~PopupDialog(void);

    // Post a message.

    void postMessage(char*,char* mesg2 = NULL,int modal = FALSE);
};

// Global dialog objects.

extern PopupDialog* theInfoDialog;
extern PopupDialog* theErrorDialog;
extern PopupDialog* theWarnDialog;

#endif
////////////////////////////////////////////////////////////////////////////
// PopupDialog.C : Dialog manager class.
////////////////////////////////////////////////////////////////////////////

#include "PopupDialog.h"
#include "Application.h"

#include <Xm/MessageB.h>

PopupDialog* theWarnDialog  = NULL;
PopupDialog* theInfoDialog  = NULL;
PopupDialog* theErrorDialog = NULL;

// Constructor creates dialog of the type needed and stores the class of dialog
// created.

PopupDialog::PopupDialog(char* name,int type):BasicWidget(name)
{
  _class  = type;
  _widget = createDialog(theApplication->toplevel());
}

PopupDialog::~PopupDialog(void)
{
  XtDestroyWidget(_widget);
}

// Create dialog.

Widget PopupDialog::createDialog(Widget parent)
{
  Widget dialog;

  switch(_class)
  {
    case INFO:

      dialog = XmCreateInformationDialog(parent,_widgetName,NULL,0);
      break;

    case ERROR:

      dialog = XmCreateErrorDialog(parent,_widgetName,NULL,0);
      break;

    case WARN:
```

```
        dialog = XmCreateWarningDialog(parent,_widgetName,NULL,0);
        break;
    }

    XmString ok    = XmStringCreateLtoR("OK",XmSTRING_DEFAULT_CHARSET);
    XmString more  = XmStringCreateLtoR("More",XmSTRING_DEFAULT_CHARSET);
    XmString title = XmStringCreateLtoR(_widgetName,XmSTRING_DEFAULT_CHARSET);

    XtVaSetValues(dialog,
                  XmNdialogTitle,     title,
                  XmNokLabelString,   ok,
                  XmNhelpLabelString, more,
                  XmNautoUnmanage,    TRUE,
                  NULL);

    XmStringFree(ok);
    XmStringFree(more);
    XmStringFree(title);

    XtUnmanageChild(XmMessageBoxGetChild(dialog,XmDIALOG_CANCEL_BUTTON));

    XtAddCallback(dialog,
                  XmNokCallback,&PopupDialog::okCallback,
                  (XtPointer)this);

    XtAddCallback(dialog,
                  XmNhelpCallback,&PopupDialog::moreCallback,
                  (XtPointer)this);

    return dialog;
}

// If dialog exists and is not in use return dialog. Otherwise create a dialog
// of the type needed and return it.

Widget PopupDialog::getDialog(void)
{
    if(_widget && !XtIsManaged(_widget))
    {
      return _widget;
    }

    Widget dialog = createDialog(theApplication->toplevel());

    return dialog;
}

// Post message or messages.

void PopupDialog::postMessage(char* mesg1,char* mesg2,int modal)
{
    Widget dialog = getDialog();

    if(modal)
    {
      XtVaSetValues(dialog,
                    XmNdialogStyle,XmDIALOG_FULL_APPLICATION_MODAL,
                    NULL);
    }

    XmString str = XmStringCreateLtoR(mesg1,XmSTRING_DEFAULT_CHARSET);

    XtVaSetValues(dialog,
                  XmNmessageString,str,
                  NULL);

    XmStringFree(str);

    if(mesg2 != NULL)
    {
      XmString moreStr = XmStringCreateLtoR(mesg2,XmSTRING_DEFAULT_CHARSET);

      XtVaSetValues(dialog,XmNuserData,moreStr,NULL);

      XtManageChild(XmMessageBoxGetChild(dialog,XmDIALOG_HELP_BUTTON));
    }
```

```
  else
  {
    XtUnmanageChild(XmMessageBoxGetChild(dialog,XmDIALOG_HELP_BUTTON));
  }

  XtManageChild(dialog);
}

void PopupDialog::okCallback(Widget dialog,XtPointer clientData,XtPointer)
{
  PopupDialog* obj = (PopupDialog*)clientData;

  if(obj->baseWidget() != dialog)
  {
    XtDestroyWidget(dialog);
  }
}

void PopupDialog::moreCallback(Widget widget,XtPointer clientData,XtPointer)
{
  PopupDialog* obj = (PopupDialog*)clientData;

  obj->moreExecute(widget);
}

void PopupDialog::moreExecute(Widget widget)
{
  XmString moreStr;
  Widget   moreDialog = createDialog(widget);

  XtVaGetValues(widget,XmNuserData,&moreStr,NULL);

  XtVaSetValues(moreDialog,
                XmNmessageString, moreStr,
                XmNautoUnmanage,  TRUE,
                NULL);

  XtUnmanageChild(XmMessageBoxGetChild(moreDialog,XmDIALOG_HELP_BUTTON));

  XtManageChild(moreDialog);
}
////////////////////////////////////////////////////////////////////////////
// PullDownMenu.h : Support a pulldown cascading menu.
////////////////////////////////////////////////////////////////////////////

#ifndef PULLDOWNMENU_H
#define PULLDOWNMENU_H

#include "CmdInterface.h"

#include <Xm/Xm.h>

// Forward reference.

class CmdList;

class PullDownMenu:public CmdInterface
{
  private:

    CmdList* _menuList;      // List of menu commands.
    Widget   _cascadeWidget;

  public:

    PullDownMenu(const char*,Widget,int,int help = FALSE);
    virtual ~PullDownMenu(void);

    // Add to menu list.

    void add(CmdInterface*);

    // Manage and unmanage menu items.

    virtual void manage(void);
    virtual void unmanage(void);
```

```
    // Activate and deactivate menu items.

    virtual void activate(void);
    virtual void deactivate(void);

    // No execute callback.

    virtual void execute(Widget,XtPointer){}
};

#endif
////////////////////////////////////////////////////////////////////////////
// PullDownMenu.C : Support a pulldown cascading menu.
////////////////////////////////////////////////////////////////////////////

#include "PullDownMenu.h"
#include "CmdList.h"

#include <Xm/RowColumn.h>
#include <Xm/CascadeB.h>

// Constructor creates a cascading widget and initializes menu list.

PullDownMenu::PullDownMenu(const char* name,Widget parent,int active,int help):
                                                          CmdInterface(name)
{
  XmString labelStr = XmStringCreateLtoR((char*)name,XmSTRING_DEFAULT_CHARSET);

  _widget = XmCreatePulldownMenu(parent,(char*)name,NULL,0);

  _cascadeWidget = XtVaCreateManagedWidget(name,
                                           xmCascadeButtonWidgetClass,
                                           parent,
                                           XmNlabelString, labelStr,
                                           XmNmnemonic,    name[0],
                                           XmNsubMenuId,   _widget,
                                           XmNsensitive,   active,
                                           XmNfontList,    _fontList,
                                           NULL);

  if(help)
  {
    XtVaSetValues(parent,
                  XmNmenuHelpWidget,_cascadeWidget,
                  NULL);
  }

  XmStringFree(labelStr);

  _menuList = new CmdList(name);
}

PullDownMenu::~PullDownMenu(void)
{
}

void PullDownMenu::add(CmdInterface* cmd)
{
  _menuList->add(cmd);
}

void PullDownMenu::activate(void)
{
  XtSetSensitive(_cascadeWidget,TRUE);
  _menuList->activate();
}

void PullDownMenu::deactivate(void)
{
  XtSetSensitive(_cascadeWidget,FALSE);
  _menuList->deactivate();
}

void PullDownMenu::manage(void)
{
  CmdInterface::manage();
  _menuList->manage();
```

```
}

void PullDownMenu::unmanage(void)
{
  CmdInterface::manage();
  _menuList->unmanage();
}
////////////////////////////////////////////////////////////////////////////
// MenuBar.h : Support a menubar.
////////////////////////////////////////////////////////////////////////////

#ifndef MENUBAR_H
#define MENUBAR_H

#include "BasicWidget.h"

// Forward references.

class CmdInterface;
class CmdList;

class MenuBar:public BasicWidget
{
  private:

    CmdList* _menuList;            // List of menu items.

  public:

    MenuBar(char*,Widget);
    ~MenuBar(void);

    // Manage and unmanage menu items.

    virtual void manage(void);
    virtual void unmanage(void);

    // Add menu items.

    void add(CmdInterface*);
};

#endif
////////////////////////////////////////////////////////////////////////////
// MenuBar.C : Support a menubar.
////////////////////////////////////////////////////////////////////////////

#include "MenuBar.h"
#include "CmdList.h"

#include <Xm/RowColumn.h>

// Constructor creates a menubar and a list of menu items.

MenuBar::MenuBar(char* name,Widget parent):BasicWidget(name)
{
  _widget   = XmCreateMenuBar(parent,"MenuBar",NULL,0);
  _menuList = new CmdList("MenuBar List");
}

MenuBar::~MenuBar(void)
{
  delete _menuList;
}

// Add commands (menu items).
void MenuBar::add(CmdInterface* cmd)
{
  _menuList->add(cmd);
}

// Manage the menubar and the list of menu items.

void MenuBar::manage(void)
{
  BasicWidget::manage();
  _menuList->manage();
```

```
}

// Unmanage the menubar and the list of menu items.

void MenuBar::unmanage(void)
{
  BasicWidget::unmanage();
  _menuList->unmanage();
}
////////////////////////////////////////////////////////////////////////////////
// MainWindow.h : Support a main window for the program.
//                Support initial screen and menubar.
////////////////////////////////////////////////////////////////////////////////

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include "BasicWindow.h"

// Forward references.

class MenuBar;
class Canvas;
class CmdList;
class CmdInterface;

class MainWindow:public BasicWindow
{
  private:

    MenuBar* _menuBar;      // Menubar for the program.
    Widget   _workArea;
    Widget   _drawArea;     // Drawing area to draw initial screen.
    Canvas*  _canvas;       // Canvas to draw initial screen.

    CmdList* _cmdList;

    // Callback functions.

    static void inputCallback(Widget,XtPointer,XtPointer);
    static void exposeCallback(Widget,XtPointer,XtPointer);

    void inputExecute(void);
    void exposeExecute(void);

    // Start the program.

    void start(void);

    // Draw the initial screen.

    void draw(void);

  public:

    MainWindow(char*);
    virtual ~MainWindow(void);

    Widget workArea(void);

    // Create the drawing area.

    virtual void initialize(void);

    // Manage main window.

    virtual void manage(void);

    // Create the main window shell and widget.

    virtual void createShell(void);
    virtual void createWidget(void);

    // Add menubar to be managed by the main window.

    void addMenuBar(MenuBar*);
    void add(CmdInterface*);
```

```
};

#endif
////////////////////////////////////////////////////////////////////////////
// MainWindow.C : Support a main window for the program.
//                Support initial screen and menubar.
////////////////////////////////////////////////////////////////////////////

#include "MainWindow.h"
#include "MenuBar.h"
#include "Canvas.h"
#include "CmdList.h"
#include "CmdInterface.h"
#include "Application.h"

#include <Xm/MainW.h>
#include <Xm/DrawingA.h>
#include <Xm/RowColumn.h>
#include <assert.h>

// Constructor doesn't create the window yet.
// Creates a list to add command buttons to be included in the main window.

MainWindow::MainWindow(char* name):BasicWindow(name,"MainWindow")
{
  _cmdList = new CmdList("Commands List");
}

MainWindow::~MainWindow(void)
{
  if(_canvas)
  {
    delete _canvas;
  }

  delete _cmdList;
}

// Create the work area and the drawing area.

void MainWindow::initialize(void)
{
  BasicWindow::initialize();

  _workArea = XtVaCreateWidget("Work Area",
                               xmRowColumnWidgetClass,
                               _widget,
                               XmNpacking,     XmPACK_TIGHT,
                               XmNorientation,XmHORIZONTAL,
                               NULL);

  _drawArea = XtVaCreateWidget("Draw Area",
                               xmDrawingAreaWidgetClass,
                               _widget,
                               XmNwidth,  1005,
                               XmNheight, 760,
                               NULL);

  XtAddCallback(_drawArea,
                XmNinputCallback,&MainWindow::inputCallback,
                (XtPointer)this);

  XtAddCallback(_drawArea,
                XmNexposeCallback,&MainWindow::exposeCallback,
                (XtPointer)this);
}

Widget MainWindow::workArea(void)
{
  return _workArea;
}

// Create the window shell.

void MainWindow::createShell(void)
{
  _window = XtVaCreatePopupShell(_windowName,
```

```
                                topLevelShellWidgetClass,
                                theApplication->toplevel(),
                                XmNtitle,      _windowName,
                                XmNwidth,      1005,
                                XmNheight,     760,
                                XmNminHeight,  75,
                                XmNminWidth,   1005,
                                NULL);
}

// Create the main window widget.

void MainWindow::createWidget(void)
{
  _widget = XtVaCreateWidget(_widgetName,
                             xmMainWindowWidgetClass,
                             _window,
                             NULL);
}

// Add menu bar.

void MainWindow::addMenuBar(MenuBar* menuBar)
{
  _menuBar = menuBar;
}

// Add command buttons.

void MainWindow::add(CmdInterface* cmd)
{
  _cmdList->add(cmd);
}

// Manage main window and the drawing area.

void MainWindow::manage(void)
{
  BasicWindow::manage();
  XtManageChild(_drawArea);

  draw();
}

// Draw the initial screen.

void MainWindow::draw(void)
{
  assert(XtIsRealized(_window));

  _canvas = new Canvas(_drawArea);

  _canvas->create();
  _canvas->clean();

  _canvas->drawString(350,150,"VISUALIZATION OF SORTING METHODS",Font17);
  _canvas->drawString(500,300,"by",Font14);
  _canvas->drawString(425,350,"Vikas Muktavaram",Font17);
  _canvas->drawString(500,375,"&",Font14);
  _canvas->drawString(420,400,"Dr. M. H. Samadzadeh",Font17);
  _canvas->drawString(400,650,"Hit any key or button to continue.",Font14);

  _canvas->copy();
}

// Input callback.

void MainWindow::inputCallback(Widget,XtPointer clientData,XtPointer)
{
  MainWindow* obj = (MainWindow*)clientData;

  obj->inputExecute();
}

// When user hits any key, initialize the program

void MainWindow::inputExecute(void)
```

```
{
  XtVaSetValues(_widget,
                XmNmenuBar, _menuBar->baseWidget(),
                NULL);

  _canvas->clean();
  _canvas->drawString(300,230,"Initializing ........",Font17);
  _canvas->copy();

  start();
}

// Manage menubar to start the program and disable callbacks and also destroy
// the drawing area which is no longer needed. Manage the work area to display
// the command buttons.

void MainWindow::start(void)
{
  _menuBar->manage();

  XtVaSetValues(_window,
                XmNwidth, 1005,
                XmNheight,75,
                NULL);

  XtRemoveCallback(_drawArea,
                   XmNinputCallback,&MainWindow::inputCallback,
                   (XtPointer)this);

  XtRemoveCallback(_drawArea,
                   XmNexposeCallback,&MainWindow::exposeCallback,
                   (XtPointer)this);

  delete _canvas;
  XtDestroyWidget(_drawArea);

  XtManageChild(_workArea);
  _cmdList->manage();
}

void MainWindow::exposeCallback(Widget,XtPointer clientData,XtPointer)
{
  MainWindow* obj = (MainWindow*)clientData;

  obj->exposeExecute();
}

void MainWindow::exposeExecute(void)
{
  assert(_canvas);
  _canvas->copy();
}
```

VITA

Vikas Muktavaram

Candidate for the Degree of

Master of Science

Thesis: VISUALIZATION OF SORTING ALGORITHMS

Major Field: Computer Science

Biographical:

Personal Data: Born in Bhongir, Andhra Pradesh, India, June 30, 1972, son of Vasantha Kumari and Parthasarathy Muktavaram.

Education: Graduated from St. Mary's Junior College, Hyderabad, Andhra Pradesh, in June 1989; received Bachelor of Technology Degree in Chemical Engineering from Osmania University College of Technology, Hyderabad, Andhra Pradesh, India, in June 1994; completed the requirements for the Master of Science Degree in Computer Science at the Computer Science Department at Oklahoma State University in December 1996.

Professional Experience: Graduate Research Assistant, Department of Chemical Engineering, Oklahoma State University, August 1994 to February 1995; Student Technical Paraprofessional, Client Services, Computing and Information Services, Oklahoma State University, February 1995 to July 1996.